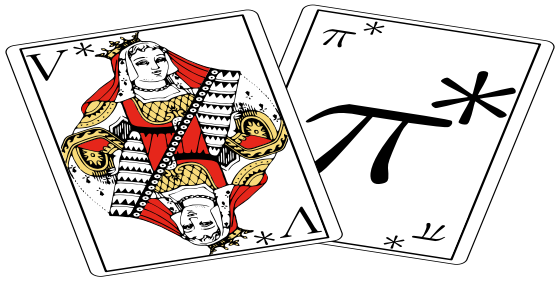
Game theory and AI: a unified approach to poker games

Thesis for graduation as Master of Artificial Intelligence

University of Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Frans Oliehoek



2 September 2005

VISIT…

Abstract

This thesis focuses on decision making in partially observable card games and, in particular, poker games. An attempt is made to outline both the game theoretic, as an agent-centric approach to such games, analyzing differences and similarities, as well as strong and weaker points and finally proposing a view to make a tradeoff between these.

The game theoretic approach for this type of games would specify a Nash-equilibrium, i.e., a pair of policies that are a best response to each other. Although a policy found in this way guarantees a minimum payoff, it is conservative in the sense that it is unable to exploit any weaknesses the opponent might have.

This motivates an agent-centric perspective, in which we propose modeling a simple poker game as a Partial Observable Markov Decision Process (POMDP) for a player who is playing against a fixed opponent whose policy is known (e.g. by repeated play). The resulting deterministic policy is a best response against the fixed opponent policy. Such a best-response policy does exploit weaknesses in the opponent's policy, thus yielding the maximum payoff attainable.

In order for the results obtained for such a simplified poker game to be of significance for real-life poker games, various methods for dealing with large (PO)MDPs are treated. These could be used to tackle larger games using the best-response approach. We examine the application of one of these methods, *model minimization*, on poker games in more detail. The result of this examination is that the reduction gained by direct application of model minimization on poker games is bounded and that this bound prevents this method from successfully tackling real-life poker variants.

Finally, in a coevolutionary framework, we try to unify the game theoretic and agent-centric approach by making a tradeoff between the security the former offers and the potential gain of the latter. A secondary goal in this approach is examining efficient calculation of Nash-equilibria.

Acknowledgments

First, I would like to thank my supervisor, Nikos Vlassis. He has been great in supporting me with his his feedback, insights and the discussions we had about them, often stretching the afternoon well into the evening. Moreover, without his high expectations and accompanying enthusiasm this thesis would have never become into what it now is.

Second, Matthijs Spaan, deserves my thanks. Especially during the first half of my graduation project he has been a big support by explaining concepts and helping out during implementation. Also, I would like to thank him for the work he put into what became my first publication.

Edwin de Jong is the last person I want to mention with name. He has been very kind and helpful in sharing his knowledge on coevolution, which resulted in chapter 7.

Finally, I'd like to thank my mother, brother, girlfriend and other friends for putting up with me during this period. They never seized to support me and my work, for which I am more than grateful.

Contents

1 Introduction **1**
- 1.1 Games . . . 1
  - 1.1.1 Why games? . . . 1
  - 1.1.2 Types of games . . . 2
  - 1.1.3 Outcomes and utilities . . . 3
- 1.2 Research on games . . . 3
- 1.3 Thesis focus . . . 4
- 1.4 Related work . . . 4
- 1.5 Two poker games . . . 5
  - 1.5.1 8-Card poker . . . 5
  - 1.5.2 Texas' Hold-em . . . 5
- 1.6 Outline of thesis . . . 6

I Games and best-response play **9**

2 Game theory **10**
- 2.1 Representation . . . 10
  - 2.1.1 Extensive form games . . . 10
  - 2.1.2 POSGs . . . 11
  - 2.1.3 Strategic form games . . . 12
  - 2.1.4 Pure policies . . . 13
- 2.2 Solutions . . . 14
  - 2.2.1 Nash equilibria . . . 14
  - 2.2.2 Solving games . . . 14
  - 2.2.3 Solving two-player zero-sum games . . . 15
  - 2.2.4 Properties of Nash equilibria . . . 18
- 2.3 The exponential gap . . . 19
  - 2.3.1 Gala language and generating the game tree . . . 19
  - 2.3.2 Sequences . . . 20
  - 2.3.3 Realization weights . . . 21
  - 2.3.4 Solving games in sequence form . . . 22
- 2.4 Remaining problems . . . 24

3 MDPs & POMDPs **25**
- 3.1 MDPs . . . 25
  - 3.1.1 The MDP framework . . . 26
  - 3.1.2 Solving MDPs . . . 27

3.2 POMDPs . . . 29
3.2.1 The POMDP framework . . . 29
3.2.2 The relation between MDP and POMDP . . . 30
3.2.3 Solving POMDPs . . . 31
3.3 From game to POMDP . . . 31
3.3.1 8-card poker as a POMDP . . . 32
3.3.2 Best-response play: Solving the POMDP . . . 33
3.3.3 Discussion . . . 34

4 Experimental results **35**
4.1 The Gala system . . . 35
4.1.1 Modifications and additions . . . 35
4.1.2 Description of resulting policy . . . 35
4.1.3 Which are optimal policies? . . . 36
4.1.4 Conclusions of verification . . . 38
4.2 Best-response play . . . 38
4.2.1 8-card poker as a POMDP . . . 39
4.2.2 Alternating learning . . . 40

II Scaling up: reduction and approximating methods 41

5 Representing large state spaces **42**
5.1 State Representation . . . 42
5.1.1 Factored representations . . . 43
5.1.2 Methods for factored MDPs . . . 45
5.1.3 Finding reduced models . . . 45
5.1.4 Other approaches . . . 46
5.2 Model Minimization . . . 47
5.2.1 Aggregation and partitions . . . 47
5.2.2 Equivalence notions . . . 47
5.2.3 The Markov property . . . 48
5.2.4 Markov requirements . . . 49
5.2.5 Computing stochastic bisimilarity . . . 51
5.2.6 Complexity and non-optimal splitting . . . 53

6 Poker & aggregation **54**
6.1 Implicit states . . . 54
6.2 Bisimilarity for poker . . . 55
6.2.1 1-action poker . . . 55
6.2.2 Optimal split for 1-action poker . . . 57
6.2.3 Bound implications . . . 59
6.3 Bisimilarity revised . . . 61
6.3.1 Uniform distributions . . . 61
6.3.2 Future research . . . 62

III Unifying winnings and security 63

7 Coevolution and security 64
7.1 Coevolution . . . 64
7.1.1 Solution concepts . . . 65
7.1.2 Memory . . . 65
7.2 Nash equilibrium solution concept . . . 65
7.2.1 Symmetric games and Nash equilibria . . . 65
7.2.2 Components of the Nash-memory . . . 66
7.2.3 The operation . . . 66
7.3 Coevolution for 8-card poker . . . 68
7.3.1 Asymmetric games . . . 68
7.3.2 Best-response heuristic . . . 69
7.3.3 The resulting algorithm . . . 70
7.4 From mixed to stochastic policies . . . 70
7.4.1 Problem and concepts . . . 70
7.4.2 Using realization weights . . . 71
7.4.3 Calculating realization weights . . . 73
7.4.4 Calculating the stochastic policy . . . 75
7.5 Experiments . . . 76
7.5.1 8-card poker . . . 76
7.5.2 Some larger poker games . . . 77
7.5.3 Security vs. best-response payoff . . . 78
7.6 Discussion . . . 79

8 Conclusions 81
8.1 Future work . . . 81

A Gala system modifications 83

Chapter 1

Introduction

Playing games is something that comes natural to humans. We easily understand the rules and by playing against more experienced players we pick up the subtleties and overcome difficulties for a particular game. In contrast, learning a computer to play a game is a considerable more difficult process.

Especially when chance moves and partial observability are involved, as is the case for games like poker, games quickly become intractable. An often used solution for this problem is to have a computer play according to some heuristics that are defined by human knowledge about a particular game. This essentially comes down to programs playing a set of predetermined rules. The major downside of this approach is that these type of programs have a very limited capability to adjust their play and, therefore, are beaten rather easily by human players or other program designed specifically to counter the heuristics behind the rules.

In this thesis we will examine frameworks that give a fundamental basis for games and are less vulnerable than rule-based programs based on human expertise.

1.1 Games

In the last century a lot of research has been devoted to the study of games. Before diving into the details of research on poker and games, we will first give a brief overview of some of this research and answer the necessary question "Why one would research games in the first place?"

1.1.1 Why games?

Probably the best reason for studying games is that games can be used to model a lot of real-life situations. Because of this, game theory has been widely applied in fields as economics, biology, (international) politics and law. Also in computer science game theory has found more and more applications. Examples of these are interface design, discourse understanding, network routing, load sharing, resource allocation in distributed systems and information and service transactions on Internet [35].

	full information	partial information
deterministic	Chess, Go	Battleships
stochastic	Backgammon, Monopoly	Poker

Table 1.1: Examples of various game types characterize by the forms of uncertainty.

This shows that games are useful for a large class of problems. Particularly most situations in which multiple interacting entities have to make decisions are suitable to be modeled as a game. In fact the interest in games has been renewed by the research in multi-agent systems.

We should mention that the by 'game' we do not mean arcade computer-games such as Doom. However, the ideas and techniques that are considered here might also be employed in certain aspects of these types of computer-games. This could also be of importance, as the computer-game industry is one of the fastest growing sectors within the entertainment branch.

Apart from their relevance games also have some properties that make them very suitable for research: Games have a set of clearly stated rules and they have a specific goal. This makes it possible to test the success of different approaches for a specific game. As an example, the research performed on chess brought many scientific advances.

1.1.2 Types of games

Games can be characterized by various properties they embody. Some important characteristics are induced by the type(s) of uncertainty present in a game [51]. One type of uncertainty is *opponent uncertainty*, meaning not knowing how your opponent will play. This is a form of uncertainty is shared by most, if not all multi-player games.

Another type of uncertainty is known as effect uncertainty: It is possible that a player does not know all possible effects of an action, e.g. opening a box in a role playing game. This type of uncertainty is not further considered as this stretches the boundary of "a set of well defined rules".

Both types of uncertainty discussed above are interesting on itself, but are less useful for characterizing games. The following two different types of uncertainty do provide important characteristics: The presence of chance moves in a game and whether the players can fully observe the current state of the game.

Chance moves are caused by the presence of *outcome uncertainty*. Outcome uncertainty occurs when all possible effects of an action and their probabilities are known, for example when throwing a dice. Games with chance moves are referred to as *stochastic games*, those without as *deterministic.*

When one or more players can't fully observe the current state of the game, the game exhibits *state uncertainty*. We say the player has partial or imperfect information regarding the state and consequently speak of partial information games.

Table 1.1 gives examples of games with the outcome and state uncertainty.

1.1.3 Outcomes and utilities

Another important factor in characterizing a game is what kind of outcomes is has. In general an outcome of a game specifies a reward for each player independently. This means that there may be outcomes that are good for all players, outcomes that are bad for all players and outcomes that are good for one, but bad for another player. This implies games can also be specified by the type of preferences the players hold over the outcomes. One such type are *strictly competitive* games: when the players in the game strictly prefer different outcomes, the game is said to be strictly competitive.

Now, lets make the idea of preference more concrete. The preferences the player holds over outcomes is expressed by a *utility function*, U. This is a mapping from outcomes to real numbers in such a way that for all outcomes o_1 and o_2 it holds that, if the player prefers o_1 over o_2, then $U(o_1) > U(o_2)$.

The utility of a certain outcome is also referred to as the *payoff*. When the payoffs for all players sum to 0, we speak of a *zero-sum* game. Clearly, a two-person zero-sum game is strictly competitive.

The games that are considered in this thesis are poker variants that have a outcomes expressed in won or lost money. The amount of money won and lost by the players sums to zero for these games.[1] However, for the game to be zero-sum, the utility payoffs should sum to one. Therefore we make the assumption that the utility function for all players is equal to the amount of money won or lost.

Also, when a game includes chance moves, the players must also have preferences over different lotteries of outcomes. Strictly spoken this requires a well-founded choice on the desired attitude towards taking risks. However, as most games typically deal with only small winnings and losings, players are usually considered risk neutral. Therefore we can simply use the expectation of these lotteries.

The issues dealt with here belong to the field of utility theory. More information can be found in [6].

1.2 Research on games

Although research on games has been mathematically formalized only relative recently, related insights can be traced back to philosophers from ancient times. As an example, at one point Socrates sketches the setting of a soldier waiting with his comrades to repulse an enemy attack. He reasons that if the battle will be won, the effort of the soldier is not needed and therefore he would better not participate, avoiding risk of injury. On the other hand it the battle will be lost, the soldiers chance of getting hurt are even higher and therefore, he should not participate in the battle in this case either. This kind of reasoning is very much related to ideas in current game theory.

In the first half of the twentieth century a lot of research was performed on games. Important contributions were made by Zermelo, von Neumann, Morgenstern and Nash and others, leading to a formalization that could be called the 'classical game theory'.

[1] Unless played in the casino, where the house takes a percentage of the pot.

With the advent of computers, again lots of games have been studied. Until the late 90's, most of the effort focused on fully observable games. An example of a fully observable game on which computer science research focused is backgammon. In 1992 TD-Gammon was introduced in [57]. The program was able to compete with the world-class player winning some games losing some others.

The most prominent, however, was the research performed on chess: the literature on chess is extensive including dedicated journals. This research resulted many advances in computer science, especially search techniques. In 1997 for the first time the world-champion at that time, Garry Kasparov, was defeated by a computer, 'Deep Blue'.

Since then more and more attention has shifted to partial information games. Poker was identified as a next 'benchmark' problem for partial information games [1, 5] and indeed more and more research has focused on poker in the last decade. We will give a brief overview in section 1.4.

1.3 Thesis focus

In this thesis, the focus will be on frameworks for learning good policies for partially observable card games, specifically poker variants. These are stochastic games. As mentioned, we assume payoffs are equal to the amount of money won or lost so that they are zero-sum and therefore strictly competitive in the two-player case.

1.4 Related work

In this section we discuss some related work on partial observable card games and poker in particular. It only gives a brief overview, as for a more detailed description quite some knowledge is required in advance.

Probably one of the first to mathematically study poker was von Neumann [58]. He created an abstract small poker game, still known as "von Neumann poker", which he studied in detail. A similar approach was taken by Kuhn [37], who studied a simplified poker game very similar to '8-card poker', which will be use as an example throughout this thesis (see section 1.5 for a description).

More recently, poker received a lot of attention from the field of computer science and artificial intelligence. The Gala system [35] provided a way to solve partial observable games, like poker, of a higher order of magnitude than was possible before. In [5, 4] a poker program called *Loki* is described that plays the game of Texas' Hold-em (also, see section 1.5) based on opponent modeling. The successor of this program, *Poki*, [3] made it to a commercial product. In [36] describes an approach based on Bayesian networks. A game theoretic approach to a medium sized poker game called *Rhode Island hold-em*, is given in [51], employing several techniques to make the size of the game manageable. A similar approach for Texas' Hold-em is given [2].

Finally, also some other partially observable card games received attention. Before 1995 a lot of research focused on bridge [1]. More recently, the game of hearts was investigated [22].

1.5 Two poker games

As we will be discussing a lot of poker variants in this thesis, we will first describe two poker variants to familiarize with some concepts. The first is a small game from literature [35] called 8-card poker. The second is a real-life poker game, used to determine the world-champion, called Texas' Hold-em.

1.5.1 8-Card poker

In this thesis we will use a simple poker variant, 8-card poker, to illustrate various concepts more clearly. An additional benefit is that the game is small enough to be solved exactly, as we will in chapter 2. 8-Card poker is played by two players: a dealer and a gambler, who both own two coins. Before the game starts, each player puts one coin to the pot, the *ante*. Then both players are dealt one card out of a deck of eight cards (1 suit, ranks 1–8).

After the players have observed their card, they are allowed to bet their remaining coin, starting with the gambler. If the gambler bets his coin, the dealer has the option to fold or call. If the dealer folds he loses the ante, and if he calls showdown follows. If the gambler does not bet, the dealer can choose to bet his coin. If the dealer does so, the gambler will have to decide whether to fold or call. If the game reaches the showdown (neither player bets or the bet is called), the player with the highest card wins the pot.

1.5.2 Texas' Hold-em

Texas' Hold-em is a real-life poker variant. In fact, it is not one particular poker variant; there are several variants of Texas' Hold-em as well. All of these are played with anywhere from two to over ten players, although we will mostly focus on the two player poker games.

The main difference between different variants of Texas' Hold-em is the amount of money that can be bet or raised. In this respect, there are limit, no-limit and pot limit games. We will discuss limit Texas' Hold-em here first. The limit version of the game specifies two amounts, with the highest amount usually being twice the lower amount, e.g. €4 / €8. The lower amount specifies the value of a single bet or raise in the first two bet-rounds, the higher amount for the last two bet-rounds.

As might be clear, bet-rounds, of which there are four in total, take a central place in Texas' Hold-em, therefore we will first describe how one bet-round is played.

In a bet-round the first player to act has 2 options: *check* and *bet*. When he checks, he doesn't place a bet, when he bets does place a bet (of €4) thereby increasing the stakes of the game. The second player has different options depending on what the first player did. If the first player checked, the second player has the same actions check and bet. If the first player bet, the second player can *fold*, *call* or *raise*. Folding means that the player gives up, causing the opponent to win.[2] When a player calls a bet, he pays enough money to the pot to match the opponent's bet. Raising means that the player calls the

[2]Technically, the first player can also fold, as can the second player after the first player checked. However, as at these point the player does not have to pay to stay in the game, this action is dominated by checking.

name	description
Royal flush	A,K,Q,J,10 of the same suit
Straight flush	five consecutive cards of the same suit
4-of-a-kind	4 cards of the same rank
full house	3-of-a-kind + one pair, e.g.: J,J,J,4,4
flush	5 cards of same suit
straight	5 consecutive cards, .e.g. 7,8,9,10,J
3-of-a-kind	3 cards of the same rank
2-pair	2 pairs, e.g. 6,6,4,4,J
pair	2 cards of same rank, e.g. 4,9,10,K,K
high-card	the highest card, e.g. 2,5,7,8,Q off-suit

Table 1.2: Hand-types for Texas' Hold-em.

opponent's bet and places a bet on top of that. In this example, with a single bet costing €4, raising comes down to placing €8 in the pot.

A bet-round is ended when no player increased the stakes of the game in his last turn, i.e. both players checked or the last bet was called. Also, there is a maximum of 4 bets, so €16 in this example, per player per bet-round.

Now the bet-round has been described, the structure of the whole game is as follows. First the players in concern pay the ante which is called the blind bet.[3] After that all players receive two private card out of a standard deck of 52 cards. This is followed by a bet round. When the first bet-round ended, three public cards are placed, face-up, on the table, this is called the *flop*. The second bet-round follows and when ended a single public card is placed on the table. This stage is called the *turn*. After the turn the third and before last bet-round starts, this means that a single bet now costs €8 and therefore a maximum of €32 per player can be bet in this round. This third bet-round is followed be a fifth and last public card placed on the table: *the river*. After the river the last bet-round is played, also with a single bet of €8.

When both players didn't fold up to this point, showdown follows and the player that has the highest combination of five cards formed using his two private cards and the table cards wins the pot.

The variants no-limit and pot-limit differ in the bets that can be placed. As suggested by the name, in no-limit poker any amount can be betted or raised. In pot-limit hold-em, the maximum bet is determined by the amount of money that is currently in the pot.

1.6 Outline of thesis

This thesis is divided in 3 parts. In the first part we discuss games and best-response play. First, game theoretic notions and solutions are introduced in chapter 2 and we identify two weak points in the outlined game theoretic approach: the incapability of exploiting weaknesses of the opponent and the practical limitation on the size of problems that can be addressed. In chapter 3 we

[3] In Texas' Hold-em only one or two, depending on the total number of players and the exact variant, pay ante.

present a method to calculate a best-response that exploits the weaknesses of the opponent. At the end of the first part we provide experimental results for both the game theoretic and best-response approach.

In the second part we discuss methods for handling bigger games using the best-response approach. In chapter 5 an overview of relevant literature is presented. For some of the discussed methods, we analyze their applicability for poker games in chapter 6.

Finally, in the last part, we examine a way of providing a tradeoff between the security of the game theoretic solution and the potential winnings of best-response play. This is done in a coevolutionary framework and discussed in chapter 7. Chapter 8 concludes and summarizes directions for future research identified throughout the thesis.

Part I

Games and best-response play

Chapter 2

Game theory

As the name implies, game theory is the traditional approach for analyzing games. It is usually divided in two parts: cooperative and non-cooperative game theory. The cooperative game theory takes a looser approach and mostly deals with bargaining problems. The non-cooperative game theory is based on exact rules for games, so that solutions can be studied in detail. As the type of games discussed in this thesis are strictly competitive, we will focus on the non-cooperative part and leave the cooperative game theory untouched.

A natural first question to ask here is what it means to solve game? In other words: *What is a solution for a game?* In general, a solution of a game is a specification for each player how to play the game in each situation that can arise. That is, it specifies the best *strategy* or *policy* for each player.[1]

In this chapter, we will first give an introduction in necessary concepts and methods from game theory. This includes different ways games can be represented, approaches for solving games and properties of these 'solutions'. Next we will describe the Gala system presented in [35] and how it can be used to solve games.

2.1 Representation

There are different types of representations for games. The most familiar of which is a representation by the rules of the game. If someone explains how to play a certain game this is the representation that would be used. The descriptions in section 1.5 are good examples.

Although such representations by rules are the easiest way to describe games, in order to perform reasoning about game dynamics and outcomes, more formal representations are needed. In this section some commonly used formal representations are discussed.

2.1.1 Extensive form games

A commonly used representation for games is the so-called *extensive form*. We can model 8-card poker as an extensive form game with partial (imperfect)

[1] In game theory the term 'strategy' is usually adopted, while AI the term 'policy' is generally used. In this thesis, we will use the term 'policy'.

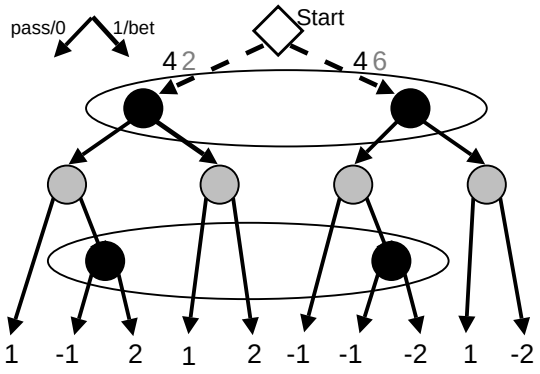


Figure 2.1: The partial game-tree of 8-card poker for the deals $(4, 2)$ and $(4, 6)$. Gambler's decision nodes are black, dealer's are grey. The diamond represent the chance move at start. The payoffs are given for the gambler.

information [38]. The extensive form of a game is given by a tree, in which nodes represent game states and whose root is the starting state. There are two types of nodes: decision nodes that represent points at which agents can make a move, and chance nodes which represent stochastic transitions 'taken by nature'. In 8-card poker, the only chance node is the starting state, in which two cards are chosen at random from the 8-card deck and are dealt to the agents.

In a partial information game, an agent may be uncertain about the true state of the game. In particular, an 8-card poker agent may not be able to discriminate between some nodes in the tree. The nodes that an agent cannot tell apart are grouped in *information sets*. From this perspective a game-tree for a perfect information game can be seen as a special case in which each node has a unique information set associated with it.

In Fig. 2.1 a part of the game-tree of 8-card poker is drawn. At the root of tree ('Start' node) a card is dealt to each agent. At each decision node the agents can choose between action 1 (*bet*), and action 0 (*fold*). The figure shows two deals: in the first the dealer receives card 2, in the second he receives card 6. The gambler receives card 4 in both cases. Therefore the gambler cannot discriminate between the two deals. This is illustrated by the information sets indicated by ovals. The leaves of the tree represent the outcomes of the game and the corresponding payoffs. In the figure only the payoff of the gambler is shown, the payoff of the dealer is exactly the opposite, as 8-card poker is a zero-sum game.

An assumption that usually is made with the analysis of extensive form games it that of *perfect recall*. This assumption in fact is not a very strong one. It embodies that at a certain node or phase in the game, the players perfectly remembers the actions he took and observations he received.

2.1.2 POSGs

As mentioned in the introduction, much of the research in multi-agent systems has renewed the interest in game theory. The framework that is often used in

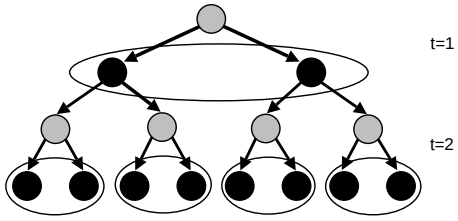


Figure 2.2: Simultaneous actions in an extensive form game. By using information sets, the first players move is hidden for the second player, modeling simultaneous actions.

this field is that of *Stochastic Games*. The partially observable variant of this is referred to as *Partially Observable Stochastic Game(POSG)* [27, 18].

POSGs are very similar to extensive form games. The major difference is that in a POSG, actions are usually taken simultaneous by all players (or agents). I.e., it specifies the space of *joint actions* A as the cross-product of the individual actions: $A = A_1 \times ... \times A_n$ for n players. As in a multi-agent environment agents usually take actions simultaneous, this framework is very natural to describe such systems. However, in an extensive form game it is also possible to model simultaneous actions, as illustrated in figure 2.2.

Another difference between the two frameworks are that in a POSG the players receive explicit observations specified by an observation model versus the implicit modeling of such observations through the use of information sets in extensive form games.

A POSG is more general than an extensive form game. The latter can be seen as a special case of the former with a tree-like structure.

2.1.3 Strategic form games

Another commonly used representation is the *strategic-* or *normal form*. A strategic form two-player game is given by a matrix and is played by a row and column player. The game is played by each player independently selecting a row/column and the outcome is given by the corresponding matrix entry.

Example 2.1.1 In table 2.1 the game of 'Chicken' is shown. The story usually told for this game concerns two teenagers who settle a dispute by driving head on at each other. Both players have the action to drive on or to chicken out. When the first player chooses to chicken out while the the second player chooses to drive on, the payoff is 0 for the first player and 2 for the second player. When both teenagers decide to drive on they will crash and therefore both receive a payoff of -1. When both player chicken out the shame is less than when only one decides to do so and both players receive a payoff of 1. □

The strategic form representation is in fact based on the notion of *pure policies*. A pure policy for a player specifies exactly one action for each situation that can occur. So rather than an action, 'chicken out' actually is a pure policy for Chicken. We will elaborate on the notion of pure policy in section 2.1.4.

	D	C
D	-1, -1	2, 0
C	0, 2	1, 1

Table 2.1: The game 'Chicken'. Both players have the option to (D)rive on or (C)hicken out.

When all players have chosen a pure policy this determines the (expected) outcome of the game.[2] This outcome is the entry in the matrix for the respective row and column corresponding to the chosen policies.

2.1.4 Pure policies

Here we will present a more precise definition of what we referred to as pure policies.

Seen from the perspective of the extensive form, a pure policy for a player specifies what action to take in each decision node for that player. Recall that in a partial information game, a player can't discriminate between the nodes within the same information set. This means that the player will have to play the same action in each of these nodes. This leads to the following definition.

Definition 2.1.1 In an extensive form game, a *pure policy*, also called *deterministic policy*, is a mapping from information sets to actions. In a strategic form game, a pure policy is a particular row or column.

As an example, in 8-card poker the dealer could follow the rule that he will always bet after receiving card 5 and having observed that the gambler passes. A collection of such rules for all combinations of cards and opponent actions would make up one pure policy.

It is possible to convert an extensive form game to one in strategic form, by enumerating all pure policies available for the players. In this transformation all information regarding the structure of the game is eliminated: the resulting normal form game only contains information regarding the outcomes. This makes it more difficult to understand what the game is about. For example it is not possible to derive who moves first from this representation. However, when only interested in which outcomes certain policies can cause, it is very suitable.

Also, it is important to see that the number of pure policies grows exponentially in the number of information sets: for each information set there are number-of-actions choices. Therefore, if n denotes the number of information sets for a player and a is the number of actions he can take at these nodes, the number of pure policies the player has is a^n. This exponential blow-up prevents methods for strategic form games to be applied to all but the simples games.

[2] When there are chance moves in the game, the expectation over the outcomes is determined.

2.2 Solutions

In this section we make the notion of solution for a game more precise. First the so-called Nash equilibria are explained. Next, some approaches to solving games are briefly reviewed. For the special case of two-player zero-sum games with partial information like poker the approach is explained in more detail.

2.2.1 Nash equilibria

The game theoretic solution of a game specifies how each player should play *given that the opponent also follows this advise*, that is it provides an optimal policy for each player. This solution of a game is given by one or more of its *Nash equilibria*.

Definition 2.2.1 Let $\pi = \langle \pi_1, \pi_2, ..., \pi_N \rangle$ be a tuple of policies for N players and let $\pi_{-k} = \langle \pi_1, ..., \pi_{k-1}, \pi_{k+1}, ..., \pi_N \rangle$ be the tuple of $N-1$ policies for player k's opponents. Also, let the expected payoff of a policy π_k for player k be given by $H_k(\pi_k, \pi_{-k})$.

A tuple of policies $\pi = \langle \pi_1, \pi_2, ..., \pi_N \rangle$ is a Nash equilibrium if and only if for all players $k = 1, ..., N$:

$$\forall_{\pi'_k} : H_k(\pi_k, \pi_{-k}) \geq H_k(\pi'_k, \pi_{-k})$$

That is, for each player k, playing π_k gives a reward equal or higher than that obtained when playing some other policy π'_k given that all other players do not deviate from their policies specified by π_{-k}. So each $\pi_k \in \pi$ is a best response for the opponents policies π_{-k}.

For example, in the Chicken in table 2.1, (C, D) is a Nash equilibrium, as chicken out is the first player's best response to the second player's policy to drive on and vice versa. Likewise, (D, C) is also a Nash equilibrium.

2.2.2 Solving games

The question to answer now is what tuple of policies to recommend as the solution. Clearly it should be a Nash equilibrium, as otherwise there would be a better policy for one of the players and he would better use that. This presents us with the question how to find a Nash equilibrium.

In extensive form games with perfect information we can find the equilibria by using Zermelo's backward induction algorithm [59]. For partial information games, however, this algorithm doesn't work because actions will have to be chosen for information sets instead of nodes. Taking a certain action in one node of the information set might give an outcome completely different than obtained when performing that same action from another node in the same information set.

For strategic form games we can use elimination of (strictly) dominated policies. For a certain player we consider if there are policies for which all the outcomes are (strictly) dominated by the outcomes for another policy. If this is the case, this policy is removed, reducing the matrix. This is repeated, iterating over the players, until no further reductions take place. Although this approach will in most cases reduce the matrix, there is absolutely no guarantee that it

will result in exactly one policy for each player. Also, when deleting non-strictly (weakly) dominated policies, equilibria may be lost.

In general, a Nash equilibrium might not exist in *pure* policies for games with partial information. We overcome this by allowing *randomized policies.* Randomized policies allow particular pure policies or actions to be played with some probability. A famous result, by Nash [40] is that for a strategic form game, there always exist at least one Nash equilibrium in randomized policies. When combining this result with the equivalence between extensive form and strategic form games [38], we obtain the following theorem:

Theorem 2.2.1 *Any extensive-form game with perfect recall has at least one Nash equilibrium in randomized policies.*

As the intuitive description above already indicated, there are two kinds of randomized policies: *mixed policies* and *stochastic policies*, which we will now define.

Definition 2.2.2 A *mixed policy,* μ, is a non-empty set of pure policies together with a probability distribution over these pure policies. The set of pure policies to which μ assigns positive probability is also called *the support of* μ.[3]

Definition 2.2.3 A *stochastic policy,* μ, is a single policy that defines a mapping from information sets to probability distributions over actions. I.e. for each information set, a stochastic policy defines what action to take with what probability.

There is a relation between mixed and stochastic policies: for every mixed policy, there is a stochastic policy that results in the same behavior and vice versa.[4] At this point, this exact relation is not important, but we will elaborate on this in chapter 7, where we show how to convert a mixed policy to a stochastic policy (7.4.2).

2.2.3 Solving two-player zero-sum games

In the previous section we briefly discussed solving games in general. Theorem 2.2.1 tells that there is at least one Nash equilibrium for every extensive form game. In general, finding such an equilibrium is difficult [44]. For two-player zero-sum games, however, things are easier.

In a zero-sum game, it is reasonable to assume that a player will try to be as harmful as possible for the opponent, because his payoff will increase as that of the opponent decreases. In the worst case an opponent will predict the players move successfully and then act to minimize the latter's payoff, thereby maximizing his own. This gives lead to playing a security or *maximin* policy.

Definition 2.2.4 Let H_1 be the payoff matrix for player 1 and let Π_1, Π_2 be the policy spaces from which respectively player 1 and player 2 can choose a policy. Then a policy π_1 that satisfies:

[3] In this thesis, policies are indicated with π in general. The notation μ is used when the policy can only be a randomized policy.

[4] This holds for games with a tree-like structure as the ones we focus on in this thesis. In general, this might not hold (e.g. in POSGs without tree-like structure).

	π_2	π_2'
π_1	-1	$+5$
π_1'	$+3$	$+2$

Table 2.2: A simple zero-sum game in strategic form with 2 policies for each player. Shown is the payoff for player 1.

$$\arg \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_2} H_1(\pi_1, \pi_2)$$

is called a *maximin* policy for player 1. The *maximin value* given by:

$$v_1 = \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_2} H_1(\pi_1, \pi_2)$$

is the payoff player 1 is guaranteed to obtain and is called the *security value* for player 1. Therefore π_1 is also called a *security policy.* Likewise, a policy π_2 that maximizes:

$$v_2 = \max_{\pi_2 \in \Pi_2} \min_{\pi_1 \in \Pi_1} H_2(\pi_1, \pi_2) \tag{2.1}$$

is a maximin policy for player 2 with payoff matrix H_2. Note that for a zero-sum game $H_1 = -H_2$ and therefore equation 2.1 can be rewritten to:

$$-v_2 = \min_{\pi_1 \in \Pi_1} \max_{\pi_2 \in \Pi_2} H_1(\pi_1, \pi_2).$$

Therefore $-v_2$ is also referred to as the *minimax value* for player 1.

We will illustrate the preceding definition with an example here.

Example 2.2.1 In table 2.2, a simple strategic form game is displayed. When player 1 assumes player 2 will predict his policy correctly, he will get -1 when playing π_1 and $+2$ when playing π_1'. His security policy is given by choosing the largest of these: π_1' giving a security payoff of $+2$, this is the maximin value for player 1.

Similarly, player 2 will get a worst-case payoff of -5 when playing π_2 and -3 when playing π_2'. Therefore player 2's security policy is π_2 with a security payoff of -3. This translates to a minimax value of $+3$ for player 1. □

In example 2.2.1 we restricted the policies that the players could pick to be pure policies. That is, we defined Π_1, Π_2 from definition 2.2.4 to be the space of pure policies. In pure policies the game has no Nash equilibrium and the security values for the players are different. Theorem 2.2.1 tells that there should be an equilibrium in randomized policies. For zero-sum games von Neumann already showed this in his minimax theorem [58]:

Theorem 2.2.2 *In a two-player zero-sum game, a policy pair π_1^*, π_2^* is in equilibrium if and only if both:*

- π_1^* *maximizes* $v_1 = \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_2} H_1(\pi_1, \pi_2)$

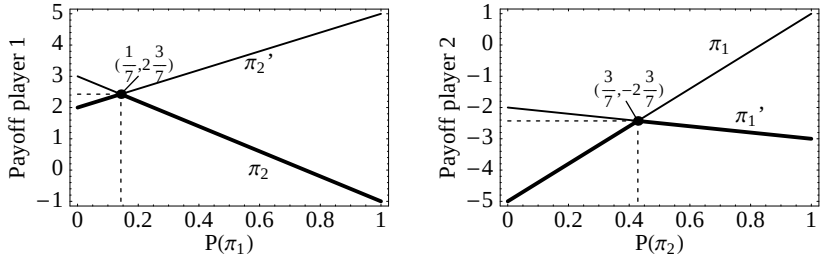


Figure 2.3: Calculating maximin values using mixed policies.

- π_2^* *maximizes* $v_2 = \max_{\pi_2 \in \Pi_2} \min_{\pi_1 \in \Pi_1} H_2(\pi_1, \pi_2)$,

where Π_1, Π_2 *are the spaces of randomized policies. In this case* $v_1 = -v_2$*, i.e. the maximin and minimax values are equal. This value is called the value of the game.*

Again, we will give an illustration of this using the example game from table 2.2.

Example 2.2.2 Let r be the probability that player 2 uses his first policy, π_2. As a consequence the probability that he uses his second policy, π_2', is $1 - r$. Now player 1 can define the expected payoff of his policies as follows:

$$\begin{aligned} E_1(\pi_1) &= r \cdot (-1) + (1 - r) \cdot 5 \\ E_1(\pi_1') &= r \cdot 3 + (1 - r) \cdot 2. \end{aligned}$$

Similarly, if t is the probability of the first player using his first policy, π_1, the expected payoff for the second player's policies is given by:

$$\begin{aligned} E_2(\pi_2) &= t \cdot 1 + (1 - t) \cdot (-3) \\ E_2(\pi_2') &= t \cdot (-5) + (1 - t) \cdot (-2). \end{aligned}$$

Also note that, because the game is zero-sum the expectation of the outcome for both players sum up to 0, i.e. $E_1(\pi_2) = -E_2(\pi_2)$, etc. This allows us to express the players' expected outcome in terms of their own policy.

Figure 2.3 graphically shows the two situations. For player 1, π_1' corresponds with $P(\pi_1) = 0$. The figure shows payoff he can expect for $t = P(\pi_1)$ against both opponent's policies. Now if player 1 assumes that player 2 will always predict his policy and act to minimize his payoff, he will get the payoff indicated by the thick line. In order to maximize this, player 1 should play his policy π_1 with a probability of 0.14 ($t = 1/7$). This is the first players security policy, obtaining a payoff of 2.42 which is the value of the game.

In a similar way, the second players security policy is playing π_2 with a probability of 0.43 ($r = 3/7$), this yields him a security level payoff of -2.42.

The pair of policies found make up a Nash-equilibrium in mixed policies. No player can increase his profit by unilaterally deviating from his current policy, so the policies are a best response to each other. □

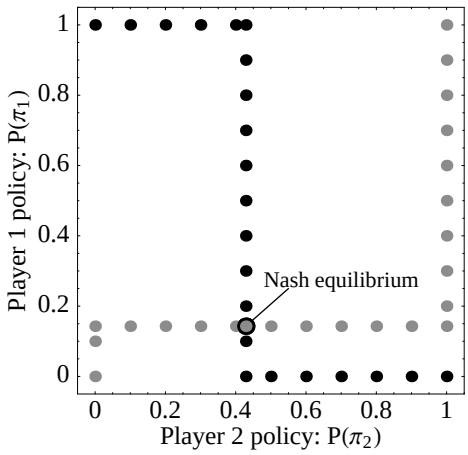


Figure 2.4: The best-response functions for the game of table 2.2. The best response function for player 1 is given in black, that for player 2 in gray. It can clearly be seen that a player is indifferent between its own policies when the opponent plays the Nash policy.

This example, of course, is very simple: both players only have two policies they can choose from. In the general case finding a solution is more difficult. However, von Neumann and Morgenstern showed [58] that for every two-player zero-sum game with a finite number of pure policies a solution can be found:

Theorem 2.2.3 *The normal form of a two-player zero-sum defines a linear program whose solutions are the Nash-equilibria of the game.*

Loosely speaking, a linear program is a maximization problem under constraints. In a normal form game the matrix, $\mathbf{A}$, gives the outcome of two pure policies played against each other. Now consider the case that the players both play a mixed policy. Let $\mathbf{x}$ denote the vector of probabilities with which the row player selects its pure policies. Similarly $\mathbf{y}$ denotes the vector of probabilities for the column player's pure policies. Then, the outcome of these mixed policies against each other is given by:

$$\mathbf{x}^T\mathbf{A}\mathbf{y}$$

The vectors $\mathbf{x}$ and $\mathbf{y}$ should both sum to 1, giving constraints. Together with the desire of both players to maximize their own payoff this can be transformed to a *linear program*, which can be solved using *linear programming*. Linear programming will be discussed in more detail in section 2.3.4.

2.2.4 Properties of Nash equilibria

As it is important to fully understand the concept Nash equilibrium, we will summarize some of the important properties that have been discussed.

- In two-player zero-sum games, a Nash policy[5] is a security policy and the value of the game is the security value for player 1.

  A security policy gives the rewards that a player can maximally obtain, given that the opponent will predict his move and act to minimize this reward. The resulting reward is the maximin or security value for the player. In general, it is paranoid to assume the opponent will do this, as other players are assumed to maximize their own rewards, not minimize that of another. In a two-player zero-sum game, however, these goals are identical.

- Nash equilibrium policies are best responses to each other.

  In fact this was how the Nash equilibrium was defined. We repeat it here to make the next point clear.

- A Nash policy is optimal *given that the opponent(s) also play a Nash policy.*

  When our opponent(s) do not play a policy from a Nash equilibrium, playing a Nash policy is still secure, but not necessarily a best-response.

- At a randomized Nash equilibrium the players are indifferent among the pure policies in the support of the Nash-policies.

  Actually this is not a property specifically for a Nash equilibrium. In general, a mixed policy is a best response to some opponent policy if and only if each of the pure policies to which is assigns positive probability is a best response to this opponent policy [6]. When this is the case, the player is indifferent between these pure policies. This is illustrated in figure 2.4.

2.3 The exponential gap

The major problem with the method outlined in 2.2.3 is the exponential blow-up when converting to strategic form. To overcome this problem Koller et al. [34] introduced a different representation called *sequence form*, that is polynomial in the size of the game tree. In [35] the Gala system was presented which makes use of this sequence form representation in order to solve games efficiently.

In this section we give an overview of the Gala system, the sequence form and exactly how to solve games using linear programming.

2.3.1 Gala language and generating the game tree

The Gala system takes as input a description of a game. This description is defined according to the Gala language and consists of definitions for: the 'name' of the game, the 'players', 'parameters' for the game, 'variables' used in the game, the 'flow' and optional modules references from within the game-flow.

The 'players' define which players participate in the game. In addition there is a special player *nature* that accounts for all the chance moves. In principle, there can be more than two players in a Gala game, but the procedure to solve a game is only implemented for the two-player (zero-sum) case.

[5] For conciseness we will refer to a policy that is part of a Nash equilibrium as a Nash policy.

'Parameters' for the game directly influence the structure of the game, for example how much stages the game does consist of, or which cards are in the deck.

'Variables' used in the game are used to maintain values through the game that for example determine the outcome or are revealed to one or more players. For example *Hand_of_player1* might be a variable in a poker game.

The 'flow' determines how the game is played. It typically invokes some modules that represent stages of the game. For example (*pay_ante*, *deal_cards*, *bet_round*) could describe the flow for a simple poker game.

From this specification the Gala system generates the game-tree by following the flow and generating nodes for each choice until the game ends. When this happens the system backs up to the last node and tries whether there was another choice available for the player to move at that node. If there is, that choice is followed, if not it backs up further. In this way the full game-tree is constructed in a depth-first manner.

2.3.2 Sequences

In order to avoid the the exponential blow-up induced when converting to normal form, the Gala system uses a different representation: the sequence form. The key observation is that pure policies result in particular paths in the game-tree, therefore distributions over pure policies induce distributions over paths, or *sequences* of moves. The probabilities of these paths can be expressed by *realization weights* and can be conveniently related to stochastic policies.

We will start with the sequences. A sequence should be interpreted as a path from the root of the game-tree to a particular node. Along this path, the edges have labels corresponding with actions and observations. To give some intuition we will first give two examples for 8-card poker: "pass on c", is a sequence for the gambler and "bet on c after seeing a pass", is one for the dealer, where c refers to observing a particular card. We give the following formal definition for a sequence:

Definition 2.3.1 A sequence $\sigma_k(p)$ for a player k is the concatenation of the *description* of the previous decision node, d_k, of that player and the action at d_k that leads to p.

The previous decision node, d_k, for player k is the first decision node of player k encountered when traversing from p to the root, excluding p itself.

The description of an decision node, d_k, is the concatenation of the labels of all edges encountered when traversing the path from root to d_k. These labels correspond with the observations and actions for player k.

By observations we mean observed actions of the opponent (e.g. 'bet', 'pass') or nature (in the form of observed cards).

Example 2.3.1 We will give some examples of sequences for gambler using figure 2.5 here. Let's take a look at node 1 and determine $\sigma_{gambler}(1)$. We first look for the previous decision node for gambler: we go up in the tree and immediately reach the root, therefore there is no previous decision node and $\sigma_{gambler}(1) = \emptyset$.

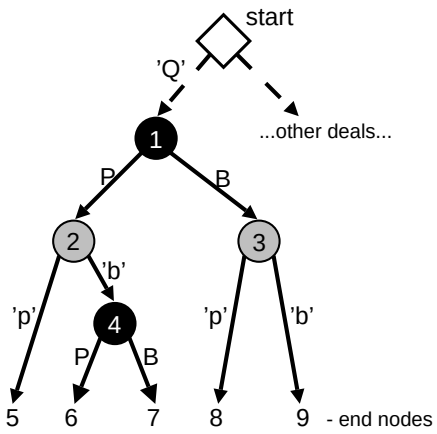


Figure 2.5: A partial game-tree for a simple poker variant from the perspective of the gambler. His actions are P(ass) and B(et). The observations gambler receives are quoted. Node 1 is some node in which the gambler received card 'Q'. 5–9 are end-nodes.

Next we examine node 4. When going up in the tree we find that the previous decision node of gambler is node 1. The description of node 1 is 'Obs(Q)'. The action taken at node 1 to reach node 4 is 'P', therefore $\sigma_{gambler}(4)$ ='Obs(Q),P'.

Node 3, 8 and 9 all have the same previous decision node; also node 1. The action taken at node 1 to reach them is also the same 'B'. Therefore $\sigma_{gambler}(3) = \sigma_{gambler}(8) = \sigma_{gambler}(9)$ ='Obs(Q),B'.

Finally for nodes 6 and 7, the previous decision node is 4. Node 4's description is 'Obs(Q),P,Obs(b)', yielding $\sigma_{gambler}(6)$ ='Obs(Q),P,Obs(b),P' and $\sigma_{gambler}(7)$ ='Obs(Q),P,Obs(b),B'. □

Note that the definition of 'description of the previous decision node' results in exactly the for player k observable labels. Therefore this description is in fact equal to the description of all the nodes in the same information set. Viewed in this way a sequence can also be seen as *the description of an information set concatenated with an action taken at that information set.*

2.3.3 Realization weights

A pure policy for player k specifies an action to take at each information set, therefore such a policy actually specifies a subset of all the nodes that can be reached when player k uses this policy. Similarly, a randomized (either stochastic or mixed) policy for player k specifies the contribution of player k in the probability that a particular node, and thus sequence, is reached or *realized.*

Now suppose we want to represent a randomized policy μ_k using sequences[6], we define the realization weights as follows:

[6]The representation of a policy using realization weights over sequences is more closely related to its stochastic representation than its mixed representation, but we keep the discussion general here.

Definition 2.3.2 The realization weight of sequence σ_k, denoted as $\mu_k(\sigma_k)$ is the probability that player k, playing according to μ_k will take the moves in σ_k, given that the appropriate information sets are reached in the game.

For example, the realization weight of the sequence 'bet on Q' in figure 2.5 is the probability the gambler bets at node 1. The realization weight of the sequence $\sigma_{gambler}(6)$: 'pass after observing a bet after passing after observing Q' is the probability of passing at node 1 times the probability of passing at node 4.

Of course not all arbitrary assignments of sequence weights represent a randomized policy. In particular, the realization weights of continuations of a sequence must sum up to the probability of that sequence. Translated to figure 2.5 this means that $\mu_{gambler}(\emptyset) = \mu_{gambler}(\sigma_{bet\,on\,Q}) + \mu_{gambler}(\sigma_{pass\,on\,Q}) = 1$, because 'bet on Q' and 'pass on Q' are continuations of the empty sequence. These constraints can be put in a constraint matrix which will be used for solving.

When all the realization weights for the set of sequences available to a player satisfy the above condition they indeed do describe a randomized policy. Therefore, when this is true for all players, a distribution over the outcomes of the game is defined. To see this, note that the realization weights give a distribution over conditional plans in the same way as the weights for full policies do in the normal form of the game.

The constraints the realization weights must obey also indicate how a realization weight representation of a policy can be converted to a stochastic policy. Let $\sigma_k(I)$ be a sequence for player k that can lead to a particular information set I. Let $\sigma_k(I) \circ a_1, ..., \sigma_k(I) \circ a_n$ be sequences that are continuations of $\sigma_k(I)$, that specify taking action $a_1, ..., a_n$ at information set I. The constraints for realization weights tell us that:

$$\mu_k(\sigma_k(I)) = \mu_k(\sigma_k(I) \circ a_1) + ... + \mu_k(\sigma_k(I) \circ a_n).$$

Therefore, when we know the realization weights of $\sigma_k(I)$ and $\sigma_k(I) \circ a_i$, the probability of taking action a_i at information set I is:

$$P(a_i|I, \mu_k) = \frac{\mu_k(\sigma_k(I) \circ a_i)}{\mu_k(\sigma_k(I))}.$$

2.3.4 Solving games in sequence form

Here a brief overview on solving sequence form using linear programming is given. For a more detailed coverage we refer to [34].

In order to solve a game we will have to formalize the outcomes over the game. For a given tuple of randomized policies $\mu = \langle \mu_1, \mu_2, ..., \mu_N \rangle$ the expected payoff H for a player is given by:

$$H(\mu) = \sum_{leaves\,p} h(p) \cdot \beta(p) \cdot \prod_{k=1}^{N} \mu_k(\sigma_k(p))$$

where $h(p)$ is the payoff the player gets at leave p, and $\beta(p)$ is the product of the probabilities of the chance moves on the path to leave p.

For two player game this can be rewritten a formulation similar to that for the normal form:

$$H(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{A} \mathbf{y}$$

where $\mathbf{x} = (x_1, x_2, ..., x_m)$ is the vector of realization weights for player 1, $\mathbf{y}$, in the same way, is the vector of realization weight for player 2. $\mathbf{A}$ is the matrix of which entry a_{ij} gives the outcome of playing σ_1^i against σ_2^j weighted by the chance moves on the path(s). That is, $\mathbf{A}$ is a matrix of which the rows correspond to the sequences for player 1 and the columns to sequences of player 2. Formally:

$$a_{ij} = \sum_{p:\sigma_1(p)=\sigma_1^i,\, \sigma_2(p)=\sigma_2^j} \beta(p) \cdot h(p).$$

Here the summation is over all p that are consistent with sequences σ_1^i and σ_2^j. Of course only leave nodes, p, will have a nonzero value for $h(p)$. Therefore the matrix $\mathbf{A}$ will have a lot of zero entries.

Now we have all the tools to define the linear program. The best response $\mathbf{y}$ to player 1's policy $\mathbf{x}$ is the following linear program:

$$\begin{aligned} \max_{\mathbf{y}} \quad & (\mathbf{x}^T \mathbf{B})\mathbf{y} \\ \textit{subject to} \quad & \mathbf{F}\mathbf{y} = \mathbf{f}, \\ & \mathbf{y} \geq 0. \end{aligned} \tag{2.2}$$

Here $\mathbf{B}$ is the payoff matrix for player 2, $\mathbf{F}$ is the constraint matrix for the assignment of realization weights $\mathbf{y}$, so they satisfy the constraints mentioned in the previous section and $\mathbf{f}$ is the column vector forcing them to add up to the right number.[7] This equation is the primal objective of the linear program. The dual objective function is:

$$\begin{aligned} \min_{\mathbf{q}} \quad & \mathbf{q}^T \mathbf{f} \\ \textit{subject to} \quad & \mathbf{q}^T \mathbf{F} \geq \mathbf{x}^T \mathbf{B}. \end{aligned} \tag{2.3}$$

Equation 2.2 and 2.3 together define the complete linear program. The optimal solution is for a pair $\mathbf{y}, \mathbf{q}$ such that the primal and dual objective are equal:

$$\mathbf{q}^T \mathbf{f} = \mathbf{q}^T \mathbf{F} \mathbf{y} = \mathbf{x}^T \mathbf{B} \mathbf{y}.$$

In a similar way the best response for player 1 can be constructed. This is optimized over a pair $\mathbf{x}, \mathbf{p}$ when:

$$\mathbf{e}^T \mathbf{p} = \mathbf{x}^T \mathbf{E}^T \mathbf{p} = \mathbf{x}^T \mathbf{A} \mathbf{y} \tag{2.4}$$

Recall that an equilibrium in a game is the point where the players' policies are best responses to each other. Therefore, we now can construct a linear program for an equilibrium for a zero-sum two player game. The primal objective function is:

[7] When performing linear programming using normal form, the constraint matrices are a single row, forcing the probability of the pure policies to sum up to 1 (i.e a scalar $\mathbf{f}$). The rest of the procedure is the same.

$$\begin{aligned} \min_{\mathbf{y},\mathbf{p}} \quad & \mathbf{e}^T\mathbf{p} \\ \textit{subject to} \quad & -\mathbf{A}\mathbf{y} + \mathbf{E}^T\mathbf{p} \geq \mathbf{0}, \\ & -\mathbf{F}\mathbf{y} = -\mathbf{f}, \\ & \mathbf{y} \geq \mathbf{0}. \end{aligned} \tag{2.5}$$

Where $\mathbf{A}$ is the payoff function for player 1, so $-\mathbf{A} = \mathbf{B}$ is the payoff function for player 2. Also in this case the program has a dual objective function, which performs a maximization over $\mathbf{q}$ and $\mathbf{x}$. The solution of the linear program gives a pair of optimal policies specified in randomization weights.

2.4 Remaining problems

In this chapter the game theoretic approach to solving games was described. We discussed what the game theoretic notion of a solution for game is and how to find such a solution. We explained how an exponential blow-up in size can be avoided by making use of sequence form instead of strategic- or normal form. The size of this sequence form is polynomial in the game-tree, allowing to tackle bigger games.

Despite all this, we argue that there are two problems with this game theoretic approach:

1. Although sequence form is polynomial in the size of the game-tree, the game-tree itself can be huge, rendering the approach less practical for real-life games.

2. The Nash equilibrium solution concept is too conservative.

The first problem is one of computation. The size of a game-tree is usually highly exponential in the size of its rule based description. As an example, for two-player Texas' Hold-em, which was discussed in the introduction, the game-tree consist of $\mathcal{O}(10^{18})$ nodes [2]. Clearly, this is a magnitude that is beyond the limits of computation.

The second problem directly relates to property discussed in section 2.2.4, that expressed that a Nash policy is optimal *given that the opponent also plays a Nash policy.* In a real-life game it is not very likely that an opponent actually plays a Nash policy. This assumption is strengthened by the first problem. In this case, we would want to exploit any weaknesses the opponent's policy might have.

This is the reason that an opponent-based approach for poker is taken in [4, 3]. It is also indicated in the setting of multi-agent systems [48]. The authors of the latter identify other problems with the usage of Nash-equilibria in [52]. In this work they also propose an 'AI Agenda' for multi-agent settings, centering around the question "*how to best represent meaningful classes of agents, and then use this representation to calculate a best response*".

Chapter 3

MDPs & POMDPs

In the previous chapter we outlined the game theoretic approach for solving games like poker and argued that its solution concept, the Nash equilibrium is too conservative for these type of games. In this chapter we switch from the field of game theory to that of *decision theoretic planning (DTP)* and artificial intelligence.

DTP studies the process of automated sequential decision making, in which the major problem is planning under uncertainty: Planning what actions to take in an uncertain environment in order to obtain the best result. This problem has been studied in various fields of science (AI planning, decision analysis, operations research, control theory, economics) and is complex. In general, the first problem is determining what 'obtaining the best result' means, usually this involves maximizing some performance measure. Luckily, for the poker-variants investigated in this thesis, this is an easy task, as this performance measure is given by the outcomes of the game.[1]

After that comes the harder task of formalizing the problem in concern and solving it such that the obtained plan or policy indeed performs well with respect to the performance measure. In this chapter, we will first introduce two frameworks, that give such a formalization for planning problems.

In section 3.1 we first introduce the *Markov Decision Process* (MDP) which has been adopted of one of the standard frameworks for planning in artificial intelligence. After that, we introduce the *Partially Observable Markov Decision Process* (POMDP) which extends the MDP.

Having explained the POMDP, in section 3.3, we show how we can convert an extensive form game to a POMDP model for a single player under the assumption of a fixed opponent, following the approach given in [42]. Finally we show how we can use this model to calculate a best-response policy that exploits the weaknesses of the opponent.

3.1 MDPs

Markov decision processes provide a formal basis to a great variety of planning problems. The basic class of problems that can be modeled using MDPs are

[1] Indeed, this is exactly one of the reasons making games suitable for research.

systems in which there is a decision maker (the *agent*) that can be modeled as stochastic processes.

An MDP planning problem is given by: 1) the possible world states, 2) the actions that can be performed at these states, 3) a transition probability model describing the probability of transferring from one particular state to another when a certain action is taken, and 4) the rewards that are assigned for certain transitions.

The goal is controlling the dynamical stochastic system the MDP describes: This system can be in one of the world states and which state changes in response to events.

One of the great advantages of the MDP framework is its ability to deal with outcome uncertainty; the uncertainty with respect of the outcome of an action. Also, it allows for modeling uncertain exogenous events, i.e. events not caused by actions of the agent, and multiple prioritized objectives. Finally, MDPs can also be used to model and solve non-terminating processes.

It is for a great part because of this versatility and flexibility, that the MDP framework has been adopted by most work on DTP and recent AI planning [8, 26, 30, 50]. Also, it has served as a basis for much work on reinforcement learning [56, 39, 50].

3.1.1 The MDP framework

Formally, a MDP is a tuple: $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, with $\mathcal{S}$ being the state-space, $\mathcal{A}$ the set of actions available to the agent, T the transition model and R the reward model. We will first elaborate on these elements of an MDP.

The state-space, $\mathcal{S}$, is the collection of world states. At each time point t the process can be in exactly one of these states $s \in \mathcal{S}$.

At each time t the agent selects an action from the set of actions that is available to him $a \in \mathcal{A}$. These actions are the only means by which the agent influences the process. Not all actions might be available in all states.

The transition model, T, specifies exactly how each action taken by the player changes the current state. Formally it is a function, $T : \mathcal{S} \times \mathcal{A} \rightarrow P(\mathcal{S}; \mathcal{S}, \mathcal{A})$, mapping from states and actions to a probability distributions over states. With some abuse of notation we will denote the probability of transitioning to s' from s when performing action a by $P(s'|s, a)$.

In its most general form, the reward model, R, specifies the reward for a particular transition. That is, is specifies a function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. Usually, however, the reward model is given as:

$$R(s, a) = \sum_{s' \in S} P(s'|s, a) \cdot R(s, a, s').$$

In some cases, the reward can also be specified as a function of only the state, giving $R(s)$. However, we will mostly use the common form $R(s, a)$, to preserve generality.

An important aspect of a MDP is that it respects the *Markov property*: the future dynamics, transitions and rewards, depend only on the current state. Formally:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1}|s_t, a_t)$$

and

$$R(s_t, a_t | s_{t-1}, a_{t-1}, ..., s_0, a_0) = R(s_t, a_t).$$

In a MDP, a policy specifies what action to take in a state, so it is a mapping from states to actions. In general, whether the MDP models a finite or infinite process is relevant for the type of policy; the last action an agent takes in its life will generally be a different one from the first action, even if the circumstances (state) are the same. The number of actions the agent takes in a MDP is called the *horizon*, h.

To model the fact that, when a MDP has a finite horizon, the preferable actions for a certain state will probably differ for different times (or *stages*), non-stationary policies are used for these type of MDPs. A non-stationary policy is a sequence of action mappings $\pi_t(s)$, with $t = 0, 1, ..., h$ [49].

For an infinite-horizon MDPs, it is known that they have an optimal stationary policy $\pi(s)$. This corresponds with the intuition that the stage will make no difference regarding what action to take at particular state.[2]

3.1.2 Solving MDPs

Now that the MDP model and the notion of policy within a MDP have been explained, we turn to the question of how we can use a MDP to solve a planning problem. It is clear that the goal is to find an optimal policy with respect to some objective function. The most common objective function is that of the *expected cumulative (discounted) reward*.

For a finite-horizon MDP of horizon h, the expected cumulative reward of a policy, π, is simply the expected value of sum of the rewards:

$$E\left[\sum_{t=1}^{h} R_t\right],$$

where R_t is the reward received at step t, which is given by:

$$R_t = \sum_{s_t \in \mathcal{S}} R(s_t, \pi_t(s_t))\, P(s_t | s_{t-1}, \pi_{t-1}(s_{t-1})).$$

For this measure to be bounded in the case of an infinite horizon MDP, a discount factor, $0 < \gamma < 1$, is introduced. The expected cumulative *discounted* reward is given by:

$$E\left[\sum_{t=1}^{\infty} \gamma^t R_t\right].$$

Now we can inductively define the value of a state according to the stationary policy π as follows:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s))\, V_\pi(s'). \tag{3.1}$$

[2] To understand why, observe that when the horizon is infinite, at each stage there are an infinite number of actions still to be taken.

For a finite horizon MDP with a non-stationary policy this definition becomes:

$$V_\pi^{t+1}(s) = R(s, \pi_t(s)) + \sum_{s'} P(s'|s, \pi_t(s))\, V_\pi^t(s'), \tag{3.2}$$

with $V_\pi^0 = 0$. Equation 3.2 defines the so-called the *t-steps-to-go value function.*

Another equation similar to the above two is:

$$V_\pi^{t+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s))\, V_\pi^t(s').$$

This equation can be used to approximate the value function for stationary policies, equation 3.1, to arbitrary accuracy, because $V_\pi^n(s) \to V_\pi(s)$ as $n \to \infty$.[3]This process is known as *successive approximation.*[9]

In the rest of this section we will focus on stationary policies. For non-stationary policies similar results hold. Also note, that non-stationary policies can be converted to stationary policies, by indexing states with their stage and requiring all transitions to go to next stage. E.g. $t' \neq t+1 \Rightarrow P(s_{t'}|s_t, a) = 0$.

Now the goal is to find an optimal policy. It is known that optimal policies share a unique optimal value function, denoted V^*. Given this optimal value function an optimal policy, π^* can be constructed greedily in the following way:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a)\, V^*(s') \right).$$

So if we can find V^* we have a way to solve the MDP. Here we discuss two ways to tackle this problem.

The first is to solve the system of Bellman equations:

$$V(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a)\, V(s') \right),$$

for all states using linear programming. [49, 14, 25]

The second option is to use dynamic programming. By iteratively applying the Bellman backup operator, H:

$$HV(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a)\, V(s') \right),$$

we can find the approximate optimal value function. In the light of non-stationary policies, the t-th application of H gives the optimal t-step-to-go value function:

$$V_{t+1}^* = HV_t^*.$$

So for a MDP with horizon k, we can apply H k times to get $(V_{t=0}^*, ..., V_{t=k}^*)$, which can be used to extract an optimal non-stationary policy. For the infinite horizon case, we are interested in the stationary policy $V^* = V_{t=\infty}^*$. Iteratively

[3]For a stationary policy, there are infinitely many steps to go.

applying H will converge to V^* in finite time. This technique is also known as *value iteration.*[56]

A different method we will not cover in detail is policy iteration. The basic idea behind this is to interleave policy evaluation (e.g. successive approximation) with policy improvement. In practice this converges in few iterations, although the amount of work to be done per iteration is more.

3.2 POMDPs

In the previous section the MDP and its ability to deal with effect uncertainty were presented. In this section the *Partially Observable Markov Decision Process* (POMDP) is described. In addition to the representational capabilities of the MDP, the POMDP model also allows for dealing with problems that exhibit *state uncertainty*, i.e. the agent does not know what the current state is, but only receives a hint regarding this true state through means of an observation.

As before we will first describe the framework. After that we will relate MDPs and POMDPs and, at the end of the section we will describe how to solve POMDPs.

3.2.1 The POMDP framework

As mentioned, in the POMDP framework the agent does not know the true state, but instead receives an observation that gives a clue regarding this state when transferring to it. To deal with this the formal description is expanded to incorporate the observations and their probabilities.

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle$, where $\mathcal{S}, \mathcal{A}, T, R$ are as before. The set $\mathcal{O}$ are the observations the agent can receive.

The observation model, O, is a function $O : \mathcal{A} \times \mathcal{S} \rightarrow P(\mathcal{O}; \mathcal{A}, \mathcal{S})$ mapping from actions and states to probability distributions over $\mathcal{O}$. We will write $P(o|a, s')$ for the probability of observation $o \in \mathcal{O}$ when transferring *to* state $s' \in \mathcal{S}$ *after* action $a \in \mathcal{A}$.

Note, that now the reward function R, can in principle also depend on the observation. However, this can again be rewritten to $R(s, a)$ in the following way:

$$R(s, a) = \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} P(s'|s, a) \cdot P(o|a, s') \cdot R(s, a, s', o).$$

As the agent can no longer observe the true state in a POMDP, a policy can't simply be a mapping from states to actions as for a MDP. Instead, at time t the agent must base his policy on the *observable history* $\langle (a_0, o_0), (a_1, o_1), ..., (a_t, o_t) \rangle$, very much like a player in an extensive form game must base its policy on his information sets.

Of course, maintaining such an history takes up a lot of space for POMDPs with a large horizon and is impossible in the case of an infinite horizon. Also, this would make the process non-Markovian. Luckily, it turns out that maintaining a probability distribution that represents the *belief* over the states provides a sufficient statistic of the history and thus a Markovian signal for the planning task.

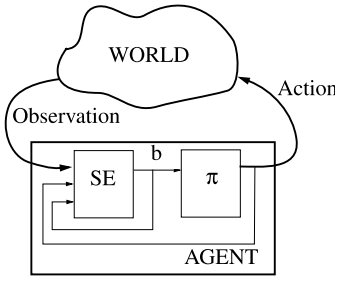


Figure 3.1: The 'state-estimator' view. (Image from [30])

A POMDP has an initial belief b_0, which is a probability distribution over the state space, with $b_0(s)$ defining the probability of starting in a state s. Every time the agent takes an action this belief is updated using Bayes' rule:

$$b_a^o(s') = \frac{P(o|s', a) \sum_{s \in \mathcal{S}} P(s'|s, a) b(s)}{P(o|a, b)}, \tag{3.3}$$

where

$$P(o|a, b) = \sum_{s' \in \mathcal{S}} P(o|s', a) \sum_{s \in \mathcal{S}} P(s'|s, a) b(s) \tag{3.4}$$

is a normalization factor.

Now, returning back to the definition of a policy, a policy in a POMDP is *a mapping from beliefs to actions.*

A nice intuitive interpretation is given by the 'state-estimator' view [12, 30], which is depicted in figure 3.1. At some point in time, the agent has a particular belief regarding the state of the world. He interacts with the world by taking an action that is based on that belief, as a consequence the state changes and the world gives back an observation. This observation is fed to the state-estimator together with the previous belief and action. The state estimator produces an updated belief which in turn is mapped to an action by the agent's policy again, etc.

3.2.2 The relation between MDP and POMDP

The MDP model as given in 3.1 sometimes is also referred to as *fully observable Markov decision process (FOMDP).* In [8] the authors explain how a FOMDP can interpreted as a special case of POMDP, namely a POMDP in which at every state the observation received is the state itself.[4]

Seen in this way, both models are part of a bigger family of MDPs. At the other end of the spectrum, there is the *non-observable MDP (NOMDP).* In this model, no observation of any kind is received. Consequently, a policy in such a model is an unconditional plan of actions.

[4]This is an idea is very similar to the view that a perfect information game can be modeled by an extensive form game in which each node has its own information set.

3.2.3 Solving POMDPs

In section 3.2.1 we saw that we could compactly represent the observable history using beliefs and that a policy in a POMDP is a mapping from these beliefs to actions. Now the question is how to find an optimal policy.

When proceeding along the same lines as before, we can define the value of a particular belief, b, under a policy π as:

$$V^{\pi}(b) = R(b,s) + \gamma \sum_{o \in \mathcal{O}} P(o|a,b) V^{\pi}(b_a^o),$$

where the reward, $R(b,s) = \sum_{s \in \mathcal{S}} R(s,a)b(s)$ [5] and the second part gives the value of all successor beliefs weighted by the probability that they will be realized when taking action a. That means that $P(o|a,b)$ is as defined in equation 3.4.

In a similar way, we can also use dynamic programming to calculate the optimal t-steps-to-go value function:

$$V_{t+1}^{*}(b) = HV_t^{*}(b),$$

where, H, the Bellman backup operator for POMDPs is given by:

$$V^{*}(b) = \max_{a \in \mathcal{A}} \left[R(b,s) + \gamma \sum_{o \in \mathcal{O}} P(o|a,b) V^{*}(b_a^o) \right]. \tag{3.5}$$

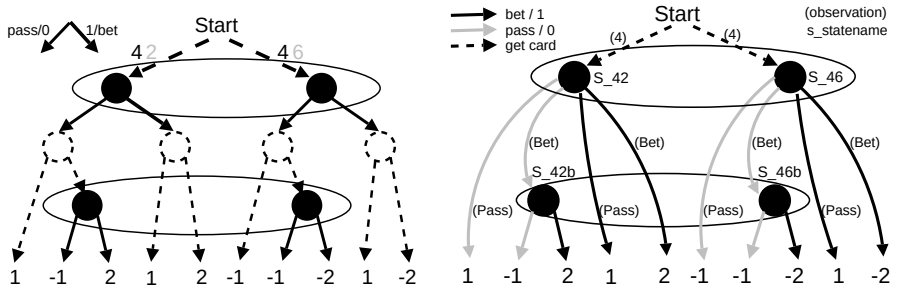
However, since beliefs are probability distributions, the belief space is continuous (a simplex with dimensionality equal to the number of states). In the general case, the optimal value over the belief space can be represented by a number of vectors (hyperplanes) that correspond to conditional plans, and the value of a belief point is given by the maximum inner product of that belief with each vector. In this way, the value function can be represented by those vectors that are maximizing for some part of the belief space. Finding those vectors is in general an intractable problem even in the finite horizon case [43], and exact algorithms are heavily relying on linear programming, [53, 11, 30].

In recent years, a lot of attention has shifted to approximate solving of POMDPs. Examples are the PEGASUS [41] algorithm which is a model-free policy search method and PERSEUS [54] which is based on randomized point based (approximate) value iteration.

3.3 From game to POMDP

Returning back to poker games, in this section we will show how we can represent such games as a POMDP and how solving a resulting POMDP yields a non-conservative policy for the protagonist agent, i.e., one that exploits the opponent.

[5] Note that at a certain belief b, $b(s)$ is the *actual* probability of state s. In this sense the word 'belief' can be slightly misleading.



(a) Extensive form game.

(b) POMDP model.

Figure 3.2: Conversion from extensive form for 8-card poker (left) to a POMDP model for the gambler (right). The decision nodes for the protagonist agent become states in the POMDP model. The deterministic choices of the opponent become stochastic transitions.

3.3.1 8-card poker as a POMDP

The crucial assumption that lies at the foundation of this approach is that the policy of the opponent is fixed and known. For example, estimated from repeated play. Given this assumption we know probability of transitioning from a particular decision node to a next decision (or outcome) node.

With this insight we can model all the decision nodes for the player in focus together with the outcome nodes as states in a POMDP. In this POMDP, the deterministic decisions of other players are converted to stochastic transitions for the protagonist agent. This is illustrated in figure 3.2, which shows a POMDP model for the gambler.

More formally, let the state-space for the POMDP, $\mathcal{S}$, consist of the set of nodes in the game-tree at which the protagonist agent select an action $a_i \in \{pass, bet\}$, including the start state[6], together with the outcome nodes, the *end-states*.

For transitions from some state in $\mathcal{S}$ to another that does not involve a move from the opponent, the transition model is clear. E.g. when the protagonist agent folds the transition is not influenced by the opponent. In the case that for a transition from s to s' an opponent move is involved, we need to consider the probabilities that he chooses his actions with.

Let $\mathcal{T}$ be the set of decision nodes for the opponent. These are all the nodes from the game-tree not in $\mathcal{S}$. At each opponent node $t \in \mathcal{T}$ he selects his action a_j according to a policy $\pi_j = P(a_j|t)$. This leads to:

$$P(s'|s, a_i) = \sum_{a_j} \sum_{t \in T} P(s'|t, a_j) P(a_j|t) P(t|s, a_i), \tag{3.6}$$

where $P(t|s, a_i)$ represents the probability induced by any chance moves before

[6]We will assume that the agent has to bet at the start node to pay the ante. In fact this is a form of 'dummy' move.

the opponent selects his action and $P(s'|t, a_j)$ that of any chance moves after the opponent selected action a_j. Also, because the transitions are over a tree, we know that each node has a unique predecessor, thus equation 3.6 reduces to:

$$P(s'|s, a_i) = P(s'|t, a_j)P(a_j|t)P(t|s, a_i).$$

In this a_j and t are exactly that action and opponent node that make s' possible, i.e. $P(s'|t, a_j) > 0$.

Having covered the construction of the transition model, we still need to define the reward- and observation model. The reward model for poker games is trivial. It is possible to use the simple version of the reward function: $R(s)$. For all the non-end-states $R(s) = 0$, the reward of the end-states is given by the corresponding outcome nodes.

The observation model also is very simple. When a player reaches a certain state he is certain to make the corresponding observation. E.g. when arriving in state s_42 in figure 3.2b, he is certain to observe card '4'.

One point of attention is that the actions of the opponent are also observations for the protagonist agent, but these remain deterministic: when the transitioning to state s_42b, the agent is certain the receive observation 'bet'. Therefore $P(o|s', a)$ is 1 for exactly one observation $o \in \mathcal{O}$.

3.3.2 Best-response play: Solving the POMDP

In section 3.2.3 we described solving POMDPs, which illustrated that this is a hard task in general. In this section we explain that for the special case of poker games this task is relatively simple.

Recall from section 3.2.1 that a belief in fact is a compressed representation of the observable history and that because of this, for an extensive form game, there is one belief per information set.

Also observe the game-tree for the discussed poker games is finite. Therefore the number of information sets and thus corresponding beliefs is finite. Moreover, the horizon of these games is relatively low and the sets $\mathcal{A}$ and $\mathcal{O}$ are relatively small, therefore the number of beliefs is not only finite, but also small. A final observation is that the initial belief is fixed and known.

To solve the resulting POMDPs, we therefore simply generate all possible beliefs and their transition probabilities, yielding a fully observable MDP. This MDP is then solved using exact value iteration as described in 3.1.

The construction of this belief MDP is straightforward. The chance of reaching a next belief is equal to the chance of receiving the observation that leads to that belief, i.e.:

$$P(b'|b, a) = P(o_i|a_i, b),$$

where a_i and o_i are the action and observation leading to belief b' and $P(o_i|a_i, b)$ is the change of receiving observation o_i after action a_i from belief b, as defined in equation 3.4.

The reward of a particular belief b is also trivially defined as:

$$R(b) = \sum_{s \in \mathcal{S}} R(s)b(s),$$

giving us the complete description of the belief MDP.

3.3.3 Discussion

Although in this thesis we focus on two-player poker games, the method for calculating a best-response policy as presented in principle works for any number of opponents. However, with a large number of players, the game-tree grows exponentially. Therefore the size of games with multiple players that can be tackled using this technique will be practically bounded.

Another remark that should be made here is that it is also possible to use the reward model that is dependent on both state and action $R(s, a)$, this eliminates the need to include end-states and end-state beliefs. As roughly half of the states are end-states this would save considerable space. In fact this should be seen as manually performing one backup step of value iteration.

A last issue is regarding our assumption of knowing the fixed opponent policy. For this assumption to be justified, it is vital to have a good opponent model. However, this is a separate topic of research and therefore not further treated in this thesis. For research on opponent modeling we refer to [4, 13, 3]. In this chapter we have shown that, given a perfect opponent model, we can calculate best-response to that policy. Of course no opponent model will be perfect in practice. We return to the issue of being more secure against errors that might come from errors in the opponent model in chapter 7.

Chapter 4

Experimental results

4.1 The Gala system

4.1.1 Modifications and additions

At the time of writing it is seven years after the Gala system was published. Therefore some modifications were needed to get everything to work. Most of the changes involved the Gala systems code. Some other modifications were necessary with respect to linear programming. These changes are described in the appendix.

Because of the required modifications, it was necessary to verify whether the Gala system indeed outputs optimal policies, as these will be used as a basis throughout this thesis. In [35] the optimal policy is given for 8-card poker, so this was used to compare to. In this section the resulting policies, a description of the comparisons made and the conclusion of the verification are given.

4.1.2 Description of resulting policy

As expected, the Gala system provided a dealer and gambler policy. These policies, however, are different from the optimal policy given in [35]. The only modification made that would seem to explain this is the usage of a different LP algorithm. This thought resulted in a second test: solving the dual of equation 2.5: which specifies optimization over the policy of the dealer (x).

This resulted in a third pair of of policies, different from both others. This strengthens the assumption that the difference is caused using a different LP algorithm: the algorithm gives different outcomes when switching the primal and dual objective function, so finding a different optimal solution than another algorithm seems more likely. The three pairs of policies are depicted in figure 4.1.

Observe that all encountered policies exhibit 'bluffing'. I.e., all specify to bet on the lowest one or two cards in some situation. In fact, bluffing is game theoretically optimal, as already shown in [35].

Another striking observation was that the value resulting from the LP optimization was +0.0625. When optimizing according to equation 2.5 , we minimize $\mathbf{e}^T\mathbf{p}$, which according to equation 2.4 is the payoff for player 1, which in the

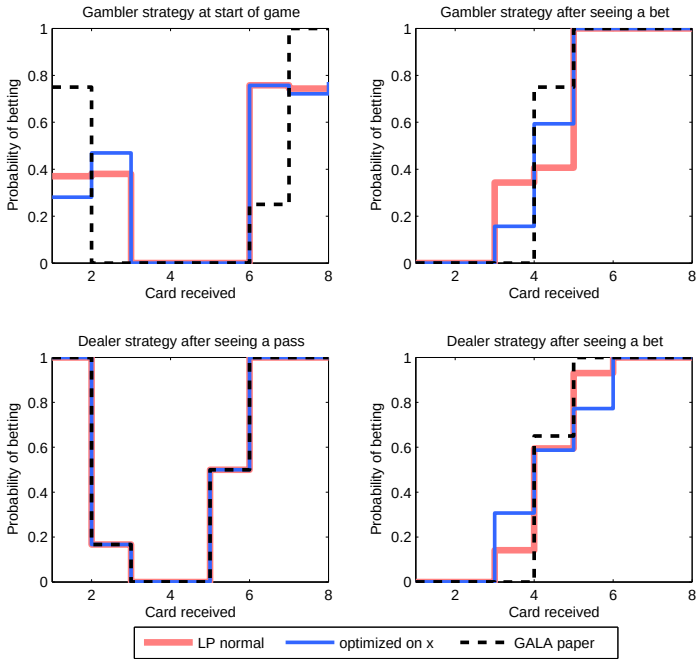

Gambler strategy at start of game
Gambler strategy after seeing a bet
Dealer strategy after seeing a pass
Dealer strategy after seeing a bet
Probability of betting
Card received
LP normal
optimized on x
GALA paper

Figure 4.1: The three resulting policies

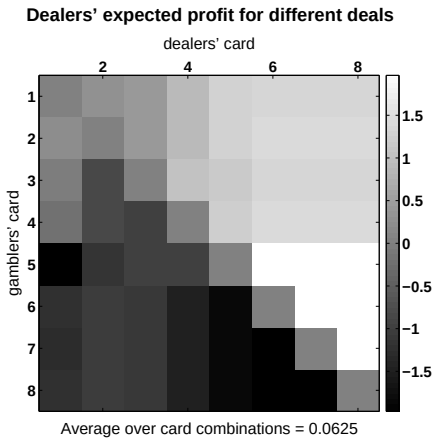
used Gala poker game is the dealer. Therefore this indicates that the value of the game, is 0.0625 coin per game in the favor of the dealer.

4.1.3 Which are optimal policies?

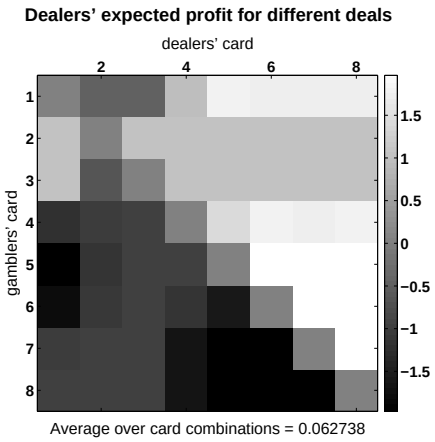
As explained in the previous section solving the 8-card poker game using the Gala system presented more questions. Out of three pairs of policies, which are optimal? And, can it be correct that the value of the game is in favor of the dealer? To answer these questions, the only viable approach seemed to do simulations. For each pair of policies five runs of a million games were simulated and the average payoff per deal was determined. By using the average outcomes of different deals, we remove the effect of some deals appearing more frequent then others, thereby influencing the average outcome.

The outcomes of these simulations are shown in figure 4.2a-c. Figure 4.2a shows the results for policies found by our modified Gala implementation using the new LP algorithm, which we will refer to as the 'LP policies'. 4.2b shows the 'Gala paper policies', i.e. those from [35]. As they were read from paper, these are quite inaccurate. Figure 4.2c shows the results for the policies that resulted from LP using the dual equation, i.e. 'optimized on x'. And finally 4.2d shows the average over all simulations.

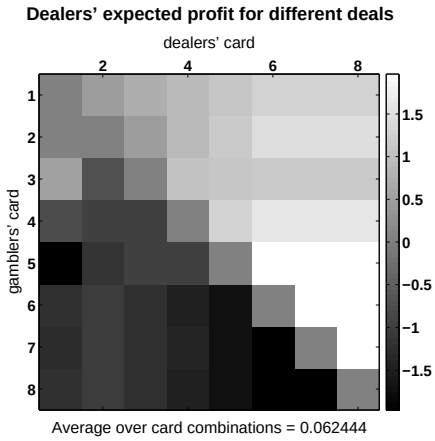
Although the average outcome for a particular deal is different for the three policy pairs, the average over these different deals lie very close together. It seems that if a combination of policies gives a higher payoff for a player for a certain deal, this is compensated by a lower payoff in the same row/column.



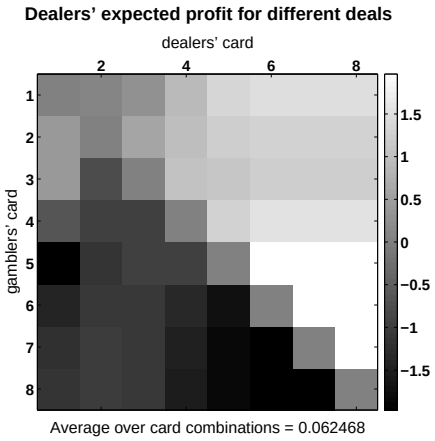
(a) LP policies.



(b) GALA paper policies.



(c) LP policies optimized on x.



(d) Average over all simulations.

Figure 4.2: Outcomes for different policy-pairs, determined by simulating 5M games

	μ	σ
LP policies	6.2500e-02	6.7685e-04
Gala paper policies	6.2738e-02	5.0112e-04
LP policies optimization on x	6.2444e-02	2.2613e-04
Gala paper dealer vs LP gambler	6.2107e-02	4.1600e-04
Gala paper gambler vs LP dealer	6.2500e-02	4.9861e-04
LP gambler vs. 'optimized on x' dealer	6.2342e-02	4.0139e-04
LP dealer 'optimized on x' gambler	6.2644e-02	7.4739e-04
Over all simulations	6.2468e-02	5.1074e-04

Table 4.1: Mean (μ) and standard deviation (σ) of expected profit for the dealer for the different simulations

For example look at the first row in figure 4.2 a and b: Although the gambler has a higher payoff for card 2 and 3 in b compared to a, this is compensated by a higher loss for cards 5-8.

The average over all deals is close to the +0.0625 coin/game predicted by the LP algorithm, for all the policy pairs. This indicates that this is the true value of the game.

Still these results didn't allow us to point one pair of policies out as being the optimal. Therefore we performed more verification by simulating games with a dealer policy selected from one pair versus a gambler from another pair. Again each simulation consisted of 5 runs of a million games. The results of this are listed in table 4.1.

As the table shows, the results are very close for all the experiments, suggesting that all policies are equally good. Moreover, the standard deviation over all simulations is not significantly higher than those within the different simulations. If some particular policies would actually be better than others, one would expect the standard deviation for that the different experiments to be lower than the over all standard deviation.

Therefore it is, in the author's opinion, safe to conclude that all the found policies are indeed optimal and that the value of the game is +0.0625 in favor of the dealer.

4.1.4 Conclusions of verification

We found that the outcomes of the different policies are close enough to justify that all are optimal. This means that the modifications, although they caused finding different optimal policies, did no harm and we conclude that we can safely use policies produced by the modified Gala implementation.

4.2 Best-response play

The procedure for calculating best-response policies as given in chapter 3 was implemented. This section describes some performed experiments.

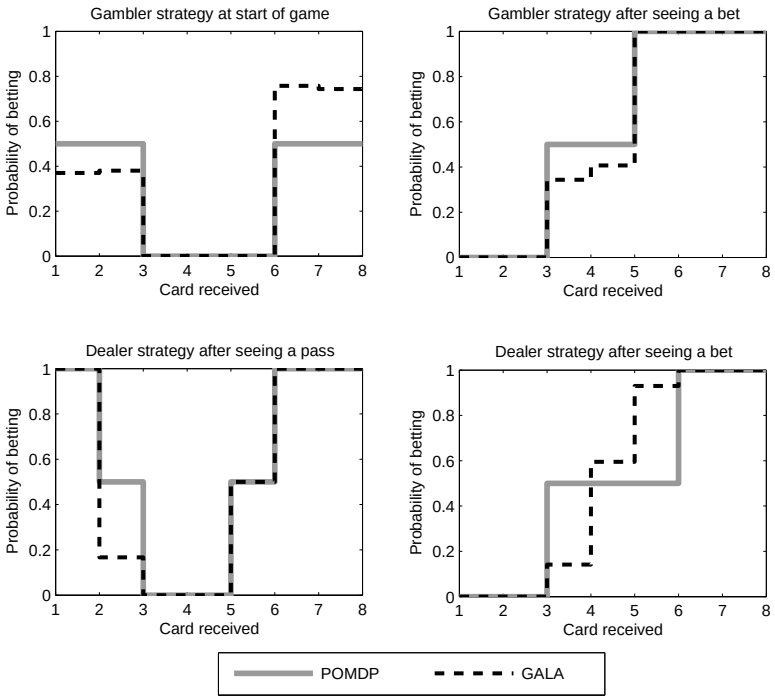


Figure 4.3: Resulting POMDP policies. Obtained when trained against Nash policies.

4.2.1 8-card poker as a POMDP

The first experiments were performed on 8-card poker. As for this game, optimal policies and the corresponding value of the game was available, this made a good test-bed for the best-response procedure.

We proceeded by calculating best-responses against the found Gala policies. As expected, the POMDP approach was able the reach a payoff of -0.0625 and $+0.0625$ for respectively gambler and dealer policies.

It turned out that when playing against the Nash-policies from Gala, there are multiple best-response policies. This is in accordance with the fact that a mixed policy is only a best response to a particular policy when all of the pure policies it assigns positive support to are best-responses, as mentioned in section 2.2.4. Figure 4.3 shows the resulting policies. For the cases that betting and passing have the same expected value (corresponding with the indifference between the different pure policies), the probability of betting is plotted as 0.5.

The figure clearly shows that when the Nash-policy specifies either bet or pass with a probability of 1.0, then so does the POMDP policy. When the Nash-policy specifies a both actions with some positive probability, the plotted POMDP policy specifies 0.5, indicating indifference. In fact the Nash and POMDP policies are very similar, only the latter is missing the particular randomization that *guarantees* the security level payoff. The lacking of this 'defen-

sive capability' becomes clear in the light of the assumption that the opponent's policy is fixed.

4.2.2 Alternating learning

After having experimentally established that the POMDP-approach to poker games indeed provides a best-response policy, we performed some experiments on *alternating learning* for 8-card poker. The idea is to start with a arbitrary policy for one of the players, learn a best response to that policy, in turn take the resulting policy and learn a best-response to that policy, etc.

It turned out that this didn't lead to any kind of convergence. This result is confirmed by theory [21], and tells us the game contains intransitive cycles.

An example of another game with such transitivities is Rock-Paper-Scissors. As rock beats scissors, scissors beats paper and paper beats rock, clearly the alternation of best-response policies will never converge.

Part II

Scaling up: reduction and approximating methods

Chapter 5

Representing large state spaces

In the previous part we showed that a partially observable card game can be transformed to a POMDP. The assumptions that were necessary are that the opponent is playing a fixed policy and that we know that fixed policy. In this setting we can exactly solve the POMDP, yielding a best-response policy.

This approach overcomes one of the identified problems a Nash equilibrium policy exhibits: being too conservative. The second problem remains. As mentioned, the POMDP representation described in section 3.3 has a state for every decision node in the game-tree belonging to the modeled player. Therefore the size of this representation is still of the same order as the full game-tree, which for realistic games is intractable.

In this chapter, we present some methods for dealing with large state spaces. First, the issue of representing state spaces for large MDPs and POMDPs is covered. After which we will focus on reducing the size of the representation through *state aggregation.* The idea is to reduce the effective size of the state space by grouping together states that are equal with respect to some equivalence notion as value and optimal action. In specific we focus on an approach called *model minimization.*

5.1 State Representation

The size of state spaces for realistic problems is the main reason that MDPs and POMDPs have not been frequently used to tackle them. As a consequence, a lot of research has focused on dealing with these large spaces, especially for the MDP framework. However, as noted in section 3.2, a POMDP can be seen as an extension of a MDP, therefore most of these methods can be extended to the POMDP framework as well.[1]

So far, we have presented the state space as an enumeration of all states, $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, this is called an *extensional* or *explicit representation.* It is also possible to describe the state space without enumerating all of them, by

[1]For conciseness, in this chapter we will often use the term MDP to denote the general family of Markov decision processes including the partial observable case.

factors	denoted	description
location	*Loc*	The robots location: (K)itchen or (O)ffice
hold coffee	*RHC*	Robot Holds Coffee?
coffee request	*CR*	Is there a unfilled coffee request?
tidy	*Tidy*	Is the office tidy?
actions	**denoted**	**description**
move	*M*	move from K to O or vice versa
pickup coffee	*PC*	pickup coffee
deliver coffee	*DC*	deliver coffee to the office
clean	*C*	make the office tidy again
events	**denoted**	**description**
mess	*Mess*	The office becomes a mess
request coffee	*Coffee!*	Someone wants: "*Coffee!*"

Table 5.1: The office robot's world

talking about properties of states or sets of states. Such representations are is called *intensional* or *implicit representation*s. Often, the full state space is thought to be the Cartesian product of several discrete properties or *factors*, for this reason the term *factored representations* is also commonly used. A big advantage of implicit representations is that can be much smaller.

Very much related to implicit representations are *abstraction* and *aggregation*. Abstraction is the process of removing properties of (particular) states that are deemed irrelevant or of little influence. The term aggregation refers to the process of grouping or aggregating states that are similar according to some equivalence notion. The resulting *aggregate states* can then be used to represent the grouped states in a reduced model.

In this section, first factored representations will be illustrated in more detail. Next, methods working directly on these factored representations will be briefly covered. After that, we will treat methods that separate model reduction from solving. In the last subsection we will mention some other approaches of dealing with large state spaces.

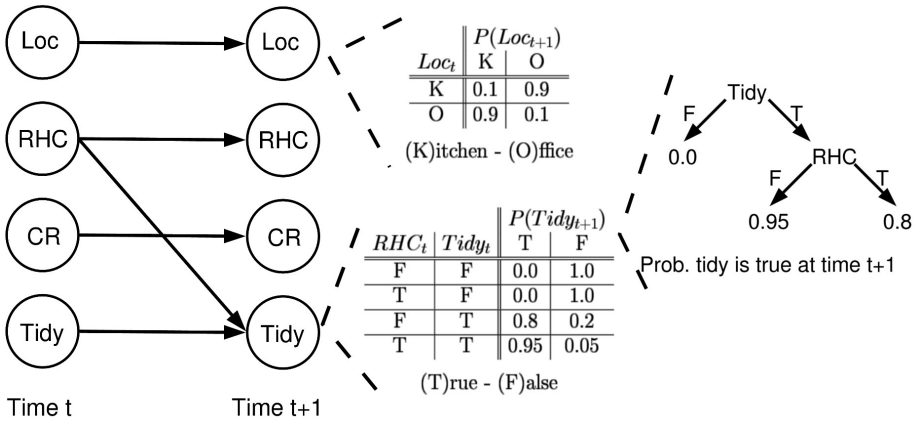
5.1.1 Factored representations

As mentioned, factored representations are based on the idea that a state can be described with some properties or factors. Let $\mathcal{F} = \{F_1, F_2, ..., F_k\}$ be the set of factors. Usually the factors are assumed to be boolean variables and easy extention to the non-boolean case is claimed.[2]

Now, a state is represented by an assignment to the k factors and the state space is formed by all possible assignments. This immediately illustrates the fact that a factored representation is typically exponentially smaller than the full state space.

Example 5.1.1 We will give a simplified example from [8] to illustrate the

[2]In the author's opinion, this extention may very well be possible, but often is not 'easy' and far from clear.



Loc_t	$P(Loc_{t+1})$ K	O
K	0.1	0.9
O	0.9	0.1

RHC_t	$Tidy_t$	$P(Tidy_{t+1})$ T	F
F	F	0.0	1.0
T	F	0.0	1.0
F	T	0.8	0.2
T	T	0.95	0.05

Figure 5.1: On the left, the 2TBN for action move, M, is shown. Also shown are the CPTs for *Loc* and *Tidy*. For *Tidy* a decision tree representation is illustrated on the right, indicating the probability *Tidy* is true after the action.

concept. Suppose we are designing a robot to help in out an office environment. Its tasks are to deliver coffee when requested and to tidy the office if it's messy.

The relevant state variables or factors are the robots location, whether it holds coffee, whether there is a coffee request and whether the office is tidy or not. Of course the robot will have several actions at its disposal: move from the kitchen to the office and vice versa, pickup and deliver coffee and clean the office.

Finally in his world there are two events that can take place, changing the state of the world: the office can become a mess and someone in the office can call for coffee. Table 5.1 summarizes 'the office robot's world'. □

In order for this presentation to be usable, we need a way to represent the transition probabilities, the rewards and, in case of partially observability, the observation probabilities. Also, we would like to find a way to do this without explicitly enumerating all the combinations.

A way of doing this is by using *two-stage temporal Bayes nets* (2TBNs) [10, 8]. A 2TBN consists of the set of factors $\mathcal{F}$ at time t and the same set at time $t+1$ and represents the influence of an action on the factors. Figure 5.1 depicts the 2TBN for the action move, *M*. The figure also depicts the *conditional probability table* (CPT) for the post-action factors *Loc* and *Tidy*. Under the action move *Loc* at time $t+1$ is only dependent on *Loc* before the action. The robot will successfully move to from the kitchen to the office (and vice versa) with a probability of 90%. The variable $Tidy$ at $t+1$ depends on two pre-action factors: $Tidy$ and RHC. When $Tidy$ is false before move, it will remain false after the move; moving does not get the office cleaner. When the office is tidy, there is a standard probability of 5% that the office becomes a mess by the people using it. However, when the robot moves while it holds coffee, there is a chance of spilling the coffee, increasing the probability of the office not being tidy after the move to 20%.

The 2TBN from this example contains no arrows between the post-action

factors. Such networks are called *simple* 2TBNs. When there are connections between the post-action factors, this means they are correlated. In such a case we speak of *general* 2TBNs. General 2TBNs require a more careful approach, for more information we refer to [7] and [8].

When using a 2TBN for each action we can fully represent the transition model compactly. Still, when the number of relevant pre-action factors increases, the CPTs grow exponentially. To counter this, the CPTs can often be represented more compactly using decision trees. For example, figure 5.1, also shows a decision tree for the CPT for *Tidy*. It illustrates that whether the robot has coffee is not relevant when it is already a mess.

As described in [29] further reduction in size can be gained by using *Algebraic Decision Diagrams* (ADDs) instead of decision trees. ADDs are an extention on ordered binary decision diagrams (OBDDs) that have been successfully applied to reduce the state space in the field of system verification. Other examples of this approach are given in [55, 8].

Up to now the explanation focused on representing the transition model in a factorized way. The extension to rewards and observation is quite simple though. For rewards we can define a *conditional reward table* (CRT) for each action. When dealing with POMDPs the same can be done for observations. In [19] these are referred to as *complete observation diagrams*. Both the rewards and observations can also be represented compactly using decision trees or ADDs.

In this section we briefly outlined factored representations based on 2TBNs. There are also other approaches such as using probabilistic STRIPS representation. For more information we refer to [8].

5.1.2 Methods for factored MDPs

Above we discussed how to compactly represent large MDPs, but we did not discuss how to solve these MDPs represented in such a way. Here we will give a brief overview of methods working directly on factored representations.

As we saw the reward function as can be represented using a decision trees or ADDs. Also note that the reward function specifies the initial value function, V_1. This has lead to various approaches that perform the bellman backup directly on these data structures. Examples are structured successive approximation (SSA) and structured value iteration (SVI). For a comprehensive overview, we refer to [8, 9].

The referred works focus on MDP, but there are also some approaches specifically for POMDPs. One example is a factored approach for POMDPs based on the incremental pruning algorithm [11] described in [28] and an approximating extension to it presented in [19].

5.1.3 Finding reduced models

In the previous subsection we mentioned some methods that solve factored MDPs directly. A different approach is to try and find a smaller model through state aggregation. This reduced model explicitly represents (enumerates) the aggregate states, which in turn implicitly represent parts of the original state space. The aggregate states correspond to a partition of the original state space. If the reduced model is small enough it can be solved exactly and will induce a policy for the original MDP.

In [16, 24] a method *model minimization* is proposed, that guarantees the optimal policy for the reduced model will induce an optimal policy for the original MDP. This approach is extended in [17, 33] to find further reduced models that induce an approximately optimal policy.

The advantage of this line of approach is that once the reduced model is constructed, we can use standard solving methods that are well understood. Also, when the parameters of the model change (but not the structure of the partition inducing the reduced model), we do not need to recalculate the reduction. Furthermore, in [24] the authors discuss equivalences between this approach and methods that operate directly on factored representations giving deeper insight in how these methods work. We will treat the model minimization method in more detail in section 5.2.

5.1.4 Other approaches

Of course, there are a lot of other approaches as well, some based on the above approaches. In [20] a non-factored approach is presented using aggregate states in the pruning phase of the incremental pruning algorithm it is based on. The method, however, does rely on an explicit representation of the full state space for performing the bellman backup.

A factored approach for POMDPs using basis functions to represent the value function is presented in [26]. It is based on the assumption of additively separable rewards, that is the assumption that different factors of the state give different components of the reward. The total reward is the sum of these components. The idea is that if rewards can be modeled additively, so can the value functions.

Another family of methods for dealing with large (PO)MDPs are based on sampling approaches. In section 3.2.3 two of these, PERSEUS [54] and PEGASUS [41] were already mentioned. The former is based on sampling belief points that are typically encountered. Then a value function and thus policy is calculated based on these belief points. The latter is based on the view that the value of a state can be approximated by sampling a small number of trajectories through the state. PEGASUS combines this perspective with policy search. Work based on a similar view is presented in [31, 32].

A final direction of recent work is that given in [46, 47]. Here the belief space is compressed in such a way that information relevant to predict expected future reward is preserved. This compression is combined with bounded policy iteration to give the VDCBPI algorithm they propose for large POMDPs.

The alternative approaches listed in this section are also relevant in the context of poker games and further research in this direction is required. Especially the trajectory sampling approaches look promising, as these guarantee performance bounds independent of the number of states. We performed a few experiments using PERSEUS for poker games, but this didn't give immediate results. Because the belief points are sampled randomly, relatively few beliefs of games reaching showdown are sampled. Further investigation along this trail might include methods that interleave sampling and policy calculation. I.e., in a subsequent iteration, beliefs are sampled using the policy from the previous iteration.

5.2 Model Minimization

Although the intuition of state aggregation is clear, formalizing it leads to the introduction of quite a few concepts. Also, care has to be taken when grouping states. In particular, when aggregating arbitrary states, the resulting aggregate states and transition model will violate the Markov property.

In this section we will first formalize state aggregation by introducing the various concepts. An important concept is that of equivalence notion. In particular, we will elaborate on the equivalence notion of *stochastic bisimilarity*, which is the central concept in *model minimization* [16, 24]. By showing that this state aggregation method preserves the Markov property, an intuition behind its working is given. After that we turn our attention on actually computing reduced models using stochastic bisimilarity and discuss some issues relevant in this procedure.

5.2.1 Aggregation and partitions

As mentioned state aggregation reduces the effective size of the state space. The result of aggregation is a partition, P, of the state space $S = \{s_1, ..., s_n\}$ that groups states together in *aggregate states* or *blocks*.[3] I.e., $P = \{B_1, B_2, ..., B_m\}$, where the blocks B_i are disjoint subsets of S. The block B_i of P to which s belongs is also denoted s/P.

A partition P' is a *refinement* of P if each block of P' is a subset of a block of P. If one of its block is a proper subset, P' is *finer* than P. The other way around, P' is called a *coarsening* of P if each block of P' is a superset of some block(s) of P and P' is *coarser* than P if it is a coarsening of P and one of its blocks is the union of some blocks in P.

In order to perform the aggregation an *equivalence notion* is used to determine what states are identical for the purposes under interest. An equivalence notion in fact is an equivalence relation, E, that induces a partition, P, of the state space: $P = S/E$. We use s/E to denote the equivalence class of s under E. This equivalence class corresponds with the block s/P.

From an equivalence relation E and its induced partition S/E, we can construct a reduced MDP. We will use M/E to denote this MDP that is defined over the aggregate states.

5.2.2 Equivalence notions

[24] first introduces two simple equivalence notions. The first is *action sequence equivalence*. Two states $i \in M$ and $j \in M'$ action sequence equivalent if and only if for all possible sequences of actions, $a_1, a_2, ..., a_n$, of any length n that start in i and j, the distribution over reward sequences, $r_1, r_2, ..., r_n$, are the the same. This also applies for two states in the same MDP, i.e. when $M = M'$.

It is also shown that this notion is inadequate as it is not able to discriminate between states with a different optimal value. This is because a policy for a MDP defines a *conditional* plan, meaning it can respond to what transitions are actually taken, while an action sequence can be seen as an unconditional plan.

[3] The term 'block' is used to refer a group of states in the partitioning process, while the term 'aggregate state' is typically used to denote a block being used as state in a reduced MDP.

This observation immediately leads to the second equivalence notion, namely *optimal value equivalence*. Under this notion, two states $s \in M$ and $t \in M'$ equivalent if they have the same optimal value. However, because the optimal value of a state does not convey information regarding the dynamics at that state, this notion is also found inadequate.

In [24] the authors pose that an adequate equivalence notion should be a refinement of both action sequence equivalence and optimal value equivalence and introduce *stochastic bisimilarity for Markov decision processes*.

Definition 5.2.1 Let $M = \langle S, A, T, R \rangle$, $M' = \langle S', A, T', R' \rangle$ be two MDPs with the same actions. let $E \subseteq S \times S'$ be a relation. E is a *stochastic bisimulation* if each $s \in S$ (and $t \in S'$) is in some pair in E, and if for all pairs $E(s,t)$, the following holds for all actions a:

1. $R(s/E)$ and $R'(t/E)$ are well defined and equal to each other.
2. For states $s' \in M$ and $t' \in M'$, such that $E(s',t')$ then $P(s'/E|s,a) = P(t'/E|s,a)$.

Two states $s \in M$ and $t \in M'$ are *stochastically bisimilar* if there is a stochastic bisimulation that relates them. Again, this definition also applies when $M = M'$ and therefore for two states in the same MDP.

In [24] the authors prove many properties of stochastic bisimulations. We will summarize some of them in the following theorem:

Theorem 5.2.1 *Stochastic bisimilarity restricted to the states of a single MDP is an equivalence relation that is a refinement of both action sequence equivalence and optimal value equivalence. Moreover, for any equivalence relation E that is a stochastic bisimulation, an optimal policy for M/E induces an optimal policy for the original MDP M.*

Proof For the proof we refer to [24]. We provide an intuition in section 5.2.3 and 5.2.4. □

5.2.3 The Markov property

This subsection shows that the Markov property may be violated when performing aggregation on arbitrary states. Assume the current state is $s \in B_i$. For a particular action a, we have that the probability of transferring to a state in B_j when performing that action is given by:

$$P(B_j|s,a) = \sum_{s' \in B_j} P(s'|s,a). \tag{5.1}$$

Let $p(\cdot)$ be a distribution over all states in B_i. We refer to this as the *within block distribution* of B_i and will also denote it $p_{B_i}(\cdot)$ if there is a need to disambiguate. This allows us to define the transition probability between B_i and B_j in the following way:

$$\begin{aligned} P(B_j|B_i, p(\cdot), a) &= \sum_{s \in B_i} p(s) \cdot P(B_j|s,a) \\ &= \sum_{s \in B_i} \sum_{s' \in B_j} p(s) \cdot P(s'|s,a). \end{aligned} \tag{5.2}$$

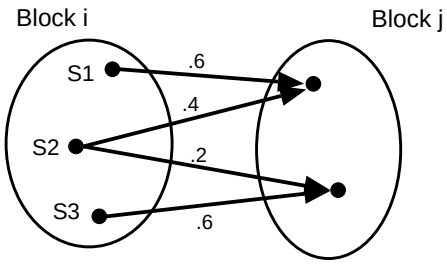


Figure 5.2: A partition satisfying equation 5.5.

This shows that the transition between blocks is depend on the distribution $p(\cdot)$. This distribution, however, can in general depend on the full history of the (PO)MDP. The result is that the transition to a next block doesn't solely depend on the current block, but potentially on the full history, breaching the Markov assumption.

For the reward[4] and observation model we can derive in a similar way:

$$R(B_i, p(\cdot), a) = \sum_{s \in B_i} p(s) \cdot R(s, a) \tag{5.3}$$

and, for a certain observation o:

$$P(o|B_i, p(\cdot), a) = \sum_{s' \in B_i} p(s') \cdot P(o|s', a). \tag{5.4}$$

Which show that the reward and observation model depend on the full history in the general case.

5.2.4 Markov requirements

After having observed that arbitrary state aggregation in general does not preserve the Markov property, we will now examine under what conditions this property *is* preserved and show how this relates to stochastic bisimilarity.

To ensure that the transition model remains Markovian, we need to ensure that for all blocks B_i, B_j and actions a the transition probability, $P(B_j|B_i, a)$, is independent of the state distribution within the blocks. A condition that will satisfy this requirement is the following:

Theorem 5.2.2 *Given a partition P. If for all $B_i, B_j \in P$ and all actions a it holds that:*

$$\forall_{s_1, s_2 \in B_i} \left(\sum_{s' \in B_j} P(s'|s_1, a) = \sum_{s' \in B_j} P(s'|s_2, a) \right), \tag{5.5}$$

[4]As mentioned in chapter 3, there are multiple ways to specify the reward and observation model: $R(s,a)$, $R(s)$, $O(o|s',a)$, $O(o|s)$. Although the POMDP models we consider can be expressed with the simpler forms and that is also used in [24], we will use the more general forms in this chapter.

then the transition model for the reduced model induced by partition P satisfies the Markov assumption.

Proof Let $P(B_j|B_i,a) \equiv \sum_{s' \in B_j} P(s'|s_1,a)$ for an arbitrary $s_1 \in B_i$. Substituting in equation 5.2 gives:

$$\begin{aligned} P(B_j|B_i,p(\cdot),a) &= P(B_j|B_i,a) \sum_{s \in B_i} p(s) \\ &= P(B_j|B_i,a) \end{aligned}$$

which is independent of the history and therefore satisfies the Markov assumption. □

The condition is illustrated in figure 5.2. Note that it is exactly this condition that is satisfied by point 2 in definition 5.2.1.

Next, we pose a condition that guarantees that the reward model that does not depend on the within block distribution and thus on the history.

Theorem 5.2.3 *If for all blocks B_i and all actions a, it holds that:*

$$\forall_{s_1,s_2 \in B_i} \quad R(s_1,a) = R(s_2,a).$$

That is, the states within all blocks have the same immediate reward with respect to all actions. Then the reward model is not dependent on the within state distribution.

Proof Let c_1 be the immediate reward for the all states in some block B_i and some action a, substitution in (5.3) gives:

$$\begin{aligned} R(B_i,p(\cdot),a) &= \sum_{s \in B_i} p(s) \cdot c_1 \\ &= c_1 \end{aligned}$$

concluding the proof. □

This says as much as "when taking an action from a state in B_i the reward is always the same, no matter what the actual state is" and corresponds with point 1 in definition 5.2.1.

The fact that definition 5.2.1 implicates theorems 5.2.3 and 5.2.3 means that a reduced MDP M/E, where E is a stochastic bisimilation, will satisfy the Markov property. This in turn implicates that any actions taken or rewards received do not depend on the history and thus provides an intuition why the action dynamics of such a reduced model are preserved and theorem 5.2.1.

Although definition 5.2.1 focuses on MDP and therefore does not mention the observations, we will also give a similar condition for the observation model. This will express as much as "when reaching a state in B_i the probability of a particular observation is fixed and doesn't depend on exactly what state is reached".

Theorem 5.2.4 *If for all blocks B_i all observations o and all actions a, it holds that:*

$$\forall_{s'_1,s'_2 \in B_i} \quad P(o|s'_1,a) = P(o|s'_2,a).$$

Then the observation model is not dependent on the within state distribution.

Proof Let c_2 be the probability $P(o|s'_1, a)$ for an arbitrary $s'_1 \in B_i$ substitution in (5.4) gives the proof in the same way as above. □

5.2.5 Computing stochastic bisimilarity

Theorem 5.2.1 tells us any stochastic bisimulation can be used to perform model reduction by aggregating the states that are equivalent under that bisimulation. The smallest model is given by the coarsest bisimulation and is referred to as the *minimal model*.

In [24] two type of approaches are given to find the coarsest bisimulation. The first type is by finding the greatest fixed point of an operator I. However, as this is done by iteratively applying $I(E)$ starting on $E_0 = S \times S$, this type of approach is infeasible for very large state spaces.

The other, more interesting approach, is based on defining a property called *stability* that can be tested locally (between blocks), but assures bisimilarity when it holds globally.

Definition 5.2.2 A block $B_i \in P$ is *stable with respect to* another block B_j if and only if for all actions a, the reward $R(B_i, a)$ is well defined and it holds that:

$$\forall_{s_1, s_2 \in B_i} P(B_j|s_1, a) = P(B_j|s_2, a).$$

A Block $B_i \in P$ is called *stable* when it is stable with respect to all blocks $B_j \in P$.

When all blocks in partition P are stable, then P is called *homogeneous*[5]. In this case, the equivalence relation E that induces this partition is also called *stable* and it is guaranteed to be a stochastic bisimulation.

Note that the formula in definition 5.2.2 is closely related to equation 5.5. The difference is that the latter additionally requires the formula to hold for all blocks $B \in P$. We therefore conclude that if a partition P is homogeneous, it will satisfy the requirement of theorem 5.2.2 and therefore the transition model of a reduced model based on this partition will not violate the Markov assumption.

The requirement that 'the reward $R(B_i, a)$ is well defined' is related to theorem 5.2.3 in the same way. Therefore, the reward model of reduced model M/E will respect the Markov assumption when the partition S/E it induces is homogeneous. In [16] a definition of stability is given that does not include the requirement on rewards. In this case, the model minimization algorithm will need to be extended to guarantee that the requirement from theorem 5.2.3 holds.

To compute the coarsest homogeneous partition and thus the minimal model, an operation $P' = SPLIT(B, C, P)$ is used. $SPLIT(B, C, P)$ takes a partition P and returns a partition P' in which block B is replaced by sub-blocks $\{B_1, ..., B_k\}$ such that all B_i are maximal sub-blocks that are stable with respect to block C.

The *model minimization algorithm* shown on the following page works by iteratively checking if there are unstable blocks and splitting them until all blocks are stable.

[5]Precisely stated, P possesses the property of *stochastic bisimulation homogeneity.*

Algorithm 1 Model minimization

```
P = {S} //trivial one-block partition
While P contains blocks B, C s.t. B is not stable w.r.t. C
  P = SPLIT(B,C,P)
end
return P //coarsest homogeneous partition
```

As mentioned, in [16] a stability notion is used that does not include any requirement on the rewards. We will call this *T-stability* to emphasize it only poses a requirement on the transition model. The version of $SPLIT$ making use of T-stability will be denoted $SPLIT$-T. We can adapt the model minimization algorithm to use $SPLIT$-T by changing the initial partition it works on.

[16] defines the *immediate reward partition,* P_{ir}, to be the coarsest partition for which the requirement of theorem 5.2.3 holds. I.e., it groups together all the states that have the same rewards for all actions. As the requirement of theorem 5.2.3 holds for the immediate reward partition, clearly it should also hold for any refinement of that partition. Also, repeated application of $SPLIT$-T on a partition P is guaranteed to yield a partition that is a refinement of P. Therefore it can be concluded that a modified version of model minimization using $SPLIT$-T applied to the immediate reward partition yields the coarsest homogeneous partition that satisfies the requirement of theorem 5.2.3.[6]

So far this section has focused on model minimization for fully observable MDPs. Now we turn our attention to partial observable MDPs. The generalization of model minimization to POMDPs given in [16, 24] is based on guaranteeing that the requirement stated in theorem 5.2.4 holds. It is done in the same way as shown above for the requirement on the reward.

Let the *observation partition*, P_o, be the coarsest partition that satisfies the requirement of theorem 5.2.4, i.e., the partition that groups together all the states that have the same observation probabilities for all actions.

Again, any refinement of the observation partition will also satisfy the requirement of theorem 5.2.4. Now let the initial partition, P, be the coarsest refinement of both the observation partition and the immediate reward partition, which we calculate as follows:

$$P = \{B_i \cap B_j \mid B_i \in P_{ir},\, B_j \in P_o\}$$

This initial partition satisfies the requirements of both theorem 5.2.3 and 5.2.4. Now performing model minimization by repeatedly applying $SPLIT$-T will result in the coarsest homogeneous partition and it will satisfy the requirements of theorems 5.2.3, 5.2.4 and 5.2.2.[7] The resulting algorithm is shown on the next page.

Another approach would be to incorporate the requirement of theorem 5.2.4 in definition 5.2.2. That is, by adding "and the observation probability $P(o|B_i, a)$

[6] When using the notion of T-stability, the notion 'homogenous' also doesn't include the requirement on the rewards within blocks anymore. (Think of 'homogeneous' as 'T-homogeneous' in this context.)

[7] The fact that it satisfies the requirement of theorem 5.2.2 follows trivially from the fact that model minimization produces a homogenous partition.

Algorithm 2 Model minimization for POMDPs

```
Pr = immediate reward partition(S)
Po = observation partition(S)
P = coarsest refinement(Pr, Po)
While P contains blocks B, C s.t. B is not T-stable w.r.t. C
  P = SPLIT(B,C,P)
end
return P //coarsest homogeneous partition
```

is well defined for all observations o and actions a". Using this definition of stable it is possible to use the normal model minimization algorithm (shown on the facing page).

5.2.6 Complexity and non-optimal splitting

The model minimization algorithm presented in the last paragraph runs in time polynomial of the resulting number of blocks, assuming that $SPLIT$ and the stability test can be computed in constant time.

Unfortunately these assumptions generally do not hold and therefore model minimization problem has been shown to be NP-hard in general. One of the problems is that to represent arbitrary partitions, blocks have to be represented as mutually inconsistent DNF formulas over the factors of the MDP. Manipulating these formulas and maintaining the shortest description of the blocks is hard. Although this complexity result seems very negative, this gives worst-case behavior. Moreover, even if finding a reduced model is costly in terms of time, it will probably still be preferable over solving the original MDP, as that might be costly in terms of space.

To reduce the cost of manipulating and maintaining block descriptions, [16, 24] introduce other block descriptions. These alternative partition representations are cheaper to manipulate, but less powerful than unconstrained DNF formulas. The result is that not all blocks and thus partitions can be represented.

To deal with this, a non-optimal splitting procedure, $SPLIT'$, is introduced. Intuitively $SPLIT'$ needs to split 'at least as much' as the optimal $SPLIT$, to guarantee a homogeneous partition as result. Formally, $SPLIT'$ is called *adequate* if $SPLIT'(B, C, P)$ is always a refinement of $SPLIT(B, C, P)$.

Model minimization making use of an adequate $SPLIT'$ operation is referred to as *adequate minimization*. Clearly, adequate minimization typically doesn't find the minimal model, because it can't represent it. From this perspective, a tradeoff is made between ease of computation and the reduction that is achieved in the resulting reduced model.

Chapter 6

Poker & aggregation

In the previous chapter various aspects of dealing with MDPs with large state spaces were covered. In this chapter we will apply some of the methods mentioned to poker games. Specifically, the theory of aggregation is related to poker games.

We show that the reduction gained by direct application of model minimization for POMDPs to poker games is bounded and argue that this approach therefore is of less practical value for these type of games. In our analysis we also identify the bottleneck and suggest a direction for further research to alleviate this problem.

6.1 Implicit states

As discussed in section 5.1 implicit or factored representations are often used to describe large states spaces. Here, we will introduce factored representations for poker games. To characterize a state in poker completely, we need to know: the sequences of actions taken, the private card(s) of both players and, if applicable, the first, second, third, etc. set of public (table) cards.

For example, for 8-card poker, we would get the following state representation:

factor	description	value
BS	bet-sequence	'01'
PC1	private card of player 1	7
PC2	private card of player 2	1

Table 6.1: Implicit state representation for 8-card poker.

which describes a state for gambler in which he observed a bet from the dealer (1) after passing (0) himself at the start of the game.

It is clear that there are some restrictions to the assignment of the variables, e.g. a state that would assign the same card to PC1 and PC2 would not correspond to a true state of the game.

Of course, the goal of an implicit representation is that it allows for reasoning about groups of states, the blocks for state aggregation, without explicitly

factor	value
BS	'01'
PC1	7-8
PC2	1-5

factor	value
BS	'01'
PC1	$7 \vee 8$
PC2	$1 \vee 5$

Table 6.2: Two ways of representing blocks for 8-card poker.

representing them. As an example, table 6.2 shows two ways we could represent blocks for 8-card poker.

While the first representation is simpler, it is less powerful as it cannot represent blocks that contain states with non-adjacent cards.

6.2 Bisimilarity for poker

In the previous section some ways of implicitly representing states and blocks were discussed. So now we will investigate how we can use the methods from chapter 5 to find a reduced model for poker games.

First, in section 6.2.1 we will introduce an example poker game called *1-action poker*. This will be used in section 6.2.2 to show that the reduction in size gained by direct application of model minimization for POMDPs as proposed in [24] is bounded. The reason is that this approach makes use of the requirement from theorem 5.2.4 on the observation probabilities as explained in section 5.2.5. We will argue that this bound is prohibitive for direct application to real-life poker variants.

6.2.1 1-action poker

Here we will introduce 1-action poker. This is also a 2-player poker variant player with a deck of 8 cards: 1–8. Both players have to pay 1 coin ante after which they receive 2 private cards each. In contrast to 8-card poker, in the betting round, the players do not have additional coins to bet. I.e the player can only do one action: check (0).

In 1-action poker, there are three 'bet'-rounds. At the end of the first two of these bet-rounds, a public card is dealt, face-up, on the table. After the third and last bet-round, the players show their cards and the player with the highest private card wins.

This game is not very entertaining, but is useful for our explanation and is closely related to real hold-em poker variants. The fact that the player with highest private card wins, means that the table cards do not influence the outcomes, but only serve as a clue.[1]

Figure 6.1 shows a part of the game-tree of 1 action poker. Indicated is that the game consists of 3 rounds, at each of which both player take one action ('check'). Because the players can only take this one action, the only branching points are the chance moves in the game. The first corresponding with the

[1] This is useful for clarity and does not affect the generality: the game could be altered to let the table cards effect the outcomes as is usual in poker (pair, straight, etc.), although this would also typically mean that the game should be played with a multi-suited deck.

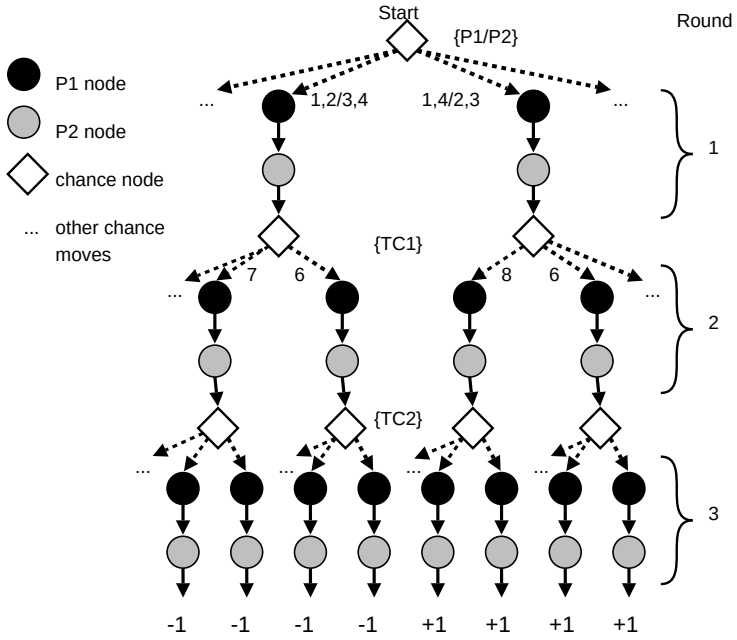


Figure 6.1: A part of the game-tree of 1 action poker.

dealing of the private cards, the ones after that corresponding with the turning of the public table cards.

Table 6.3 shows a factored state representation for a POMDP for the first player (gambler) in 1-action poker. The valid values for factors PC1, PC2, TC1 and TC2 are in fact subsets of the deck, as is indicated by the brackets.

factor	description
BS	the bet-sequence
PC1	private cards player 1
PC2	" 2
TC1	table card before round 2
TC2	" 3

factor	value
BS	'00'
PC1	{7,1}
PC2	{5,4}
TC1	{3}
TC2	-

Table 6.3: The implicit representation for 1-action poker and an example state for player 1 (gambler). BS '00' means both players played action '0', therefore it is the first player's move again. At this point TC1 is revealed and round 2 starts. TC2 is unassigned at this phase of the game.

6.2.2 Optimal split for 1-action poker

Here we will use 1-action poker to make some claims on output produced by the model minimization algorithm as presented in section 5.2.5.[2] The focus will be on the influence of requirement from theorem 5.2.4 on the observation probabilities.

Lemma 6.2.1 *When model minimization for POMDPs applied to a POMDP for 1-action poker results in partition P. Then it holds that, for all blocks $B_i \in P$ and for all states $s_1, s_2 \in B_i$, $BS(s_1) = BS(s_2)$*

Proof Notice that, there are only three bet-sequences in the game at which a player takes an action. Let's call these bs_i with $1 \leq i \leq 3$. Because there is only one action the players can take, the bet-sequence changes deterministically. Also, all end-states have the same bet-sequence ('00,00,00'), which we will call bs_{end}.

Now, suppose, $BS(s_1) \neq BS(s_2)$. That means that the bet-sequence of one of the states has more steps to go to reach bs_{end}. Let's denote this $BS(s_1) > BS(s_2)$ and assume it holds. In this case there are two possibilities: either 1) $BS(s_2) = bs_{end}$ or 2) it is not.

In case 1) s_2 is an end-state and s_1 is not. This means that $R(s_1) \neq R(s_2)$, however this is in contradiction with the result that model minimization calculates a homogeneous partition P.

In case 2) $BS(s_1)$ and $BS(s_2)$ have deterministic successors: $BS(s_1')$ and $BS(s_2')$ and it holds that $BS(s_1') > BS(s_2')$. Again, there are two cases (s_2' is an end-state and s_1' is not), inductively giving that s_1' and s_2' cannot be in the same block. This in turn gives that block B_i is not stable, again contradicting the result that model minimization calculates a homogeneous partition P. □

Intuitively, this lemma means that all blocks in the partition resulting from model minimization are 'located within the bet-rounds'.

Definition 6.2.1 The *assigned cards* specified by a state, s, is the set

$$AC(s) = PC1 \cup PC2 \cup TC1 \cup TC2.$$

Lemma 6.2.2 *When model minimization for POMDPs applied to a POMDP for 1-action poker results in partition P. Then it holds, for all blocks $B_i \in P$ and for all states $s_1, s_2 \in B_i$, that:*

1. *For all observations, o, $P(o|s_1) = P(o|s_2)$.*
2. *If block B_i is not located in the last bet-round, then $AC(s_1) = AC(s_2)$.*

Proof 1. Follows trivially from the fact the model minimization for POMDPs satisfies the requirement from theorem 5.2.4. In the remainder we prove 2.

Suppose $AC(s_1) \neq AC(s_2)$. Since B_i is not located in the last bet-round, there will be another card observation. Now let $c_1 \in AC(s_1) \backslash AC(s_2)$ be a card assigned by s_1 but not by s_2 and o_1 be the observation of that card. This means that there is a transition from s_2 to a state $s_2' \in B_j$ such that $P(o_1|s_2') > 0$. For s_1 there is not such a transition, because:

[2] Note that because there is only 1 action we will omit requirements 'for all actions' and use $R(s)$ and $P(o|s)$ in most of this explanation.

- by 1. $P(o_1|B_j) > 0$ for all states in B_j, and
- as s_1 already assigns card c_1, there is no state s'_1 it can transfer to such that $P(o_1|s'_1) > 0$.

Therefore $0 = P(B_j|s_1) \neq P(B_j|s_2) > 0$. Again, this contradicts that P is a homogeneous partition. □

Using the lemmas above, we will show that the requirement on the observation model for use in optimal SPLIT for POMDPs severely limits the maximal obtainable reduction.

To show this we first need to define some more concepts. First, let a round be a stage in the game as indicated in figure 6.1. Round i, indicates that there are $i-1$ actions taken by the protagonist agent. As seen already, because there is only one action, a round is associated with a specific bet-sequence. E.g. round 0 corresponds with the start of the game and round 4 with the end of the game: bet-sequence '00,00,00'. Also, let the d be the size of the deck. Then we can define:

- $n_{ac}(i)$ - the number of assigned cards at round i.
- $n_{oc}(i)$ - number of observed cards when reaching round i.
- $n_{uac}(i) = \frac{d!}{(d-n_{ac}(i))!\, n_{ac}(i)!}$ - denotes the number of unique assigned card combinations at round i. This is the number of unique $n_{ac}(i)$ subsets of a d-element set. E.g. at round 2 a total of five cards have been assigned ($2 \cdot 2$ private cards, plus one table card). So $n_{uac}(2) = \frac{8!}{3!5!} = 56$.
- $n_{spuac}(i)$ - the number of states per unique card combination at round i . As there is only one bet-sequence per round, this is the number of ways the $n_{uac}(i)$ cards can be assigned. E.g. $n_{spuac}(2) = \frac{5!}{2!2!1!} = 30$.
- $s(i) = n_{uac}(i) \cdot n_{spuac}(i)$ - the number of states at round i.
- $n_{opuac}(i) = \frac{n_{ac}(i)}{(n_{ac}(i)-n_{oc}(i))!n_{oc}(i)!}$ - the number of possible observations per unique assigned card combination when reaching round i. E.g. when reaching round 2 there are $n_{uac}(2) = 56$ unique card combinations, and they assign $n_{ac}(2) = 5$ cards. When reaching one of these, we just observed a single table card ($= n_{oc}(2)$), so we could have 5 observations.

Theorem 6.2.1 *Let $b(i)$ be the number of blocks, resulting from model minimization for POMDPs as given in section 5.2.5, that lie within round i. Also let $n_{opuac}(i)$ and $n_{spuac}(i)$ be as defined above. Then for $i \leq 2$, i.e. blocks not located in the last bet-round,*

$$b_{min}(i) = n_{uac}(i) \cdot n_{opuac}(i)$$

is a lower-bound for $b(i)$ in 1-action poker.

As a consequence, the reduction in size obtainable for states in these rounds is also bounded by:

$$\frac{b(i)}{s(i)} \geq \frac{b_{min}(i)}{s(i)} = \frac{n_{opuac}(i)}{n_{spuac}(i)}.$$

round, i	$n_{oc}(i)$	$n_{ac}(i)$	$n_{uac}(i)$	$n_{spuac}(i)$	$n_{opuac}(i)$	$\frac{n_{opuac}(i)}{n_{spuac}(i)}$
1	2	4	70	6	6	1
2	1	5	56	30	5	0.167
3	1	6	28	180	6	n/a

Table 6.4: Lower bounds for the maximal reduction obtainable with model minimization for POMDPs per round. The quantities relevant for determining this are also shown. For round 3, since it's the last round, the bound does not apply.

Proof We have a couple of things to prove. First we need to prove that all blocks lie within the bet-rounds, so that $b(i)$ is well-defined. This follows from lemma 6.2.1 together with the observation that each bet-sequence determines the round.

Next, we need to show that $b(i)$ is bounded by $b_{min}(i)$ for $i \leq 2$. From lemma 6.2.2 it follows that each block B_i must assign the same cards to all states it clusters. Therefore there must be at least $n_{uac}(i)$ blocks. From the same lemma it follows that the observations for all states in a block must be equal. Therefore, $b_{min}(i) = n_{uac}(i) \cdot n_{opuac}(i)$ must be a lower bound for $b(i)$.

Finally, we need to observe that:

$$\frac{b_{min}(i)}{s(i)} = \frac{n_{uac}(i) \cdot n_{opuac}(i)}{n_{uac}(i) \cdot n_{spuac}(i)} = \frac{n_{opuac}(i)}{n_{spuac}(i)},$$

immediately giving the bound on obtainable reduction. □

Table 6.4 shows the maximal reduction obtainable per round for 1-action poker and the involved quantities. A striking observation is that for the first round no reduction is obtained at all. This can be explained by noticing that for all states in a set of states that assign the same cards, the observation received is different. This is also illustrated in figure 6.1.

6.2.3 Bound implications

In the previous section a lower bound on the maximally obtainable compression using model minimization for POMDPs as presented in section 5.2.5 was derived for 1-action poker. For this derivation, only the requirement on the observation model as specified by theorem 5.2.4 was considered. The actual reduction will be lower as also the requirement on the reward model must be satisfied.[3]

Now we will argue that this bound indicates that the presented method of model minimization for POMDPs is not suitable for real-life poker variants. We will consider Texas' Hold-em as an example here. Starting with an analysis of the similarities and differences between 1-action poker with respect to the derivation.

Lemma 6.2.2 is does not depend on the action dynamics of the game in concern, therefore it is directly applicable to Texas' Hold-em.

[3]The requirement on the transition model is trivially satisfied.

i	$n_{oc}(i)$	$n_{ac}(i)$	$n_{uac}(i)$	$n_{spuac}(i)$	$n_{opuac}(i)$
1	2	4	$\frac{52!}{48!4!} = 2.65 \cdot 10^5$	$\frac{4!}{2!2!} = 6$	$\frac{4!}{2!2!} = 6$
2	3	7	$\frac{52!}{45!7!} = 1.33 \cdot 10^8$	$\frac{7!}{2!2!3!} = 210$	$\frac{7!}{4!3!} = 35$
3	1	8	$\frac{52!}{44!8!} = 7.52 \cdot 10^8$	$\frac{8!}{2!2!3!1!} = 1680$	$\frac{8!}{7!1!} = 8$
4	1	9	$\frac{52!}{43!9!} = 3.679 \cdot 10^9$	$\frac{9!}{2!2!3!1!1!} = 15,120$	$\frac{9!}{8!1!} = 9$

Table 6.5: The relevant quantities for deriving a lower bound on obtained compression applied to Texas' Hold-em under assumptions as explained in the text. i denotes the round.

In contrast lemma 6.2.1 is not directly applicable, it is very well possible that two states with a different bet-sequence are stochastic bisimilar. For example, consider a two states in the last bet-round that are equal in all respects (assigned cards, money in the pot, etc.) except for the bet-sequence in the first round. In this particular case it is possible, even likely, that our opponent will act the same in these two states, inducing the same state dynamics. Therefore these states can be stochastic bisimilar, even though the (full) bet-sequences are different.

Also, the number of states per unique assigned card combination for round i, $n_{spuac}(i)$, is larger. This is because there are 19 bet-sequences starting in the first round, giving multiple[4] states in the first round for the same card assignment. Nine out of these 19 bet-sequences transfer to the next round. This mean that in round two there are a total of $9 \cdot 19 = 171$ bet-sequences, etc.

It is clear that an analysis similar to that of 1-action poker would become very complex for Texas' Hold-em. Therefore we make the following assumption: we treat the game as if there is only one state per unique card assignment per round.[5] This means that within each round we collapse all the states that differ only on their bet-sequence into one state. It should be clear that, in general, not all these states can be collapsed in such a way while still producing a reduced model inducing an optimal policy. E.g. this would suggest that, for a state in which the opponent has raised at all occasions and another state in which he only called, the optimal action is the same. In fact this suggests that the opponent behavior specifies no information whatsoever and therefore would only be correct for an opponent playing a uniform random policy.

Now we argue that even with this assumption, that is clearly over-estimating the possible reduction, direct application of model minimization for POMDPs still presents a bound on the obtainable reduction.

This is supported by table 6.5, which displays the relevant quantities based on the assumption of one state per unique card assignment per round. As an example, the maximum obtainable reduction for round 3 is $\frac{8}{1680} \approx 0.005$. Although this seems like a big reduction, the minimum number of blocks becomes $7.52 \cdot 10^8 \cdot 8 \approx 6.02 \cdot 10^9$, which is still is impractical for computation.

[4]The exact number is 15 states for both players: 10 outcome nodes and 5 decision nodes.

[5]Note that this differs from $n_{spuac}(i)$ as this latter notion does not differentiate between states that assign a particular card to a different player (or to one of the sets of table cards).

6.3 Bisimilarity revised

As shown in section 5.2.3 aggregation of arbitrary states generally does not preserve the Markov property, because the within block distribution can be dependent on the full history. After that, conditions were posed on the transitions, observations and rewards for states within blocks such that this within block distribution becomes irrelevant. As a result, the blocks and thus aggregate states possess the Markov property. This gave the intuition behind why a stochastic bisimulation induces a reduced model that can be used to calculate an optimal policy for the original POMDP. Unfortunately, as shown in this chapter, the requirement on the observation model puts a bound on the obtainable reduction, that makes application for real-life poker games impractical.

6.3.1 Uniform distributions

Another approach would be not to pose conditions on the transitions, observations and rewards directly, but on the within block distributions itself. In other words, the condition now is that the within block distribution, $p(\cdot)$, is not dependent on the full history.

An obvious way to accomplish this is to require that for all blocks $p(\cdot)$ is always uniform.[6]

Theorem 6.3.1 *When a block B_i, for which the within state distribution $p(\cdot)$ is uniform, is used as aggregate state, this state possesses the Markov property.*

Proof We can simply replace $p(s)$ by $\frac{1}{|B_i|}$ in equations 5.2, 5.3 and 5.4, giving:

$$\begin{aligned} P(B_j|B_i,p(\cdot),a) &= \frac{1}{|B_i|}\sum_{s\in B_i}\sum_{s'\in B_j} P(s'|s,a) \\ R(B_i,p(\cdot),a) &= \frac{1}{|B_i|}\sum_{s\in B_i} R(s,a) \\ P(o|B_i,p(\cdot),a) &= \frac{1}{|B_i|}\sum_{s'\in B_i} P(o|s',a). \end{aligned}$$

All of these are history independent, concluding the proof. □

As a consequence, a partition for which all blocks satisfy the requirement of uniform within block distribution, yields a reduced Markovian model. Of course, guaranteeing this uniformity can in general be hard, but in very structured and specifically tree-like MDPs as described in this thesis this can be easier.

Figure 6.2 depicts the problem. We want to guarantee that $p_{B_j}(\cdot)$ for block B_j is uniform, i.e., $p_{B_j}(s_1) = p_{B_j}(s_2) = ... = \frac{1}{|B_i|}$. A condition that jumps to mind is that for all states $s_j \in B_j$ it should hold that $\sum_{s\in S} P(s_j|s,a)$ is equal under all actions. This condition, however, is insufficient: it does not take into account that the probabilities for reaching the predecessor states $s \in S$ can be different. Moreover, in general these probabilities can change over time.[7]

[6] Actually, the requiring that $p(\cdot)$, the within block distribution for the blocks B_i is only fixed (not uniform), is enough. However, for clearness and ease of explanation we will assume uniform within block distributions.

[7] As an example, consider the case that state s_1 in figure 6.2 links back to one of its predecessors.

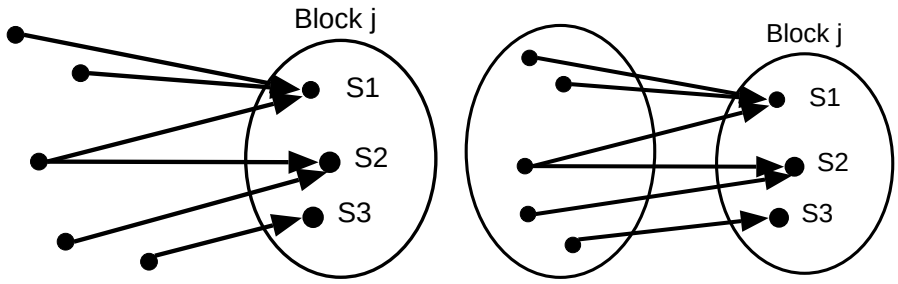


Figure 6.2: The problem of guaranteeing a uniform within block distribution. Left shows arbitrary predecessors. Right shows predecessors to be in the same block, allowing to guarantee a uniform distribution for block j.

In order to guarantee that $p_{B_j}(\cdot)$ is uniform, we pose the requirement that all states $s \in S$ that can transition to B_j are in the *same* block B_i with $p_{B_i}(\cdot)$ uniform and $\sum_{s \in B_i} P(s_j|s,a) = c$, for all $s_j \in B_j$ and some constant c.

In general, the requirement that a block has one unique predecessor block can limit the applicability. For the special case where an MDP has a tree structure, however, this requirement is less of a burden, because nodes in a tree have at most one predecessor.

6.3.2 Future research

It is not trivially clear that, when changing the requirements as posed section 5.2.4 to the requirement specified in the previous section, the resulting reduced MDP will still induce an optimal policy for the original MDP.

In fact it might be very well possible that the original constraints on the transition and reward model will need to be maintained. It is intuitively plausible, however, that the constraint on the observation model from theorem 5.2.4 may be dropped when, at the same time, the constraint specifying fixed within block distributions is satisfied. This is because the actual dynamics of the POMDP are not influenced by the observation model; observations only provide information regarding what the true state is. Trying to prove this intuition would be a first step for future work.

Of course, even if it is proven that it is possible to abandon the limiting observation constraint, there might be other bounds that limit model minimization's applicability for real-life problems. This would be a second step for research.

Part III

Unifying winnings and security

Chapter 7

Coevolution and security

We have focused on best-response against a fixed opponent given that we know how he plays. I.e., we assumed we had a perfect opponent model. Of course this is, in general, not the case, which could make our calculated policy vulnerable.

In this chapter we will discuss coevolution. This technique can be used to find policies that are more secure against multiple opponent policies. The general idea is to find a policy that is secure against a certain group or *population* of opponent policies, then to evolve that population and find a new policy that is secure against the new population. By repeating this procedure, the final policy will be secure against all opponent policies; converging to a Nash equilibrium.

The objective of this investigation is twofold. On one hand it describes an alternative way of calculating a Nash-equilibrium. Although the two-player zero-sum case can be solved in polynomial time using linear programming as described in chapter 2, for large problems this remains expensive.

On the other hand, it tries to provide a way to compromise between security and best-response payoff, thus unifying the game- and decision theoretic perspectives.

7.1 Coevolution

The idea behind evolutionary algorithms is that there is population of individuals that represent candidate solutions. By evaluating these candidates against one or more tests their fitness is determined and the fittest produce the next generation. Coevolutionary methods differ from evolutionary methods in the way they treat the tests. Instead of having one evolving population of candidate solutions and a fixed set of tests, two evolving populations are maintained: one for the candidate solution and one for the tests.

In the poker games discussed in this thesis, the population of candidate solutions could consist of a number of policies for the gambler, in which case the corresponding tests would be a set of policies for the dealer. How well one of the gambler policies performs is measured by the outcome achieved against all the tests.

7.1.1 Solution concepts

For coevolution to have any meaning it must specify a goal or *solution concept.* This can be expressed as a set of candidate solutions satisfying some requirements.

Formally, let $\mathcal{C}$ and $\mathcal{T}$ be the sets of respectively all possible candidate solutions and tests. The the outcome of a particular candidate $C \in \mathcal{C}$ against a test $T \in \mathcal{T}$ is given by the *interaction function* or *game*, $G : C \times T \to \mathbb{R}$. In the presence of chance moves in this game $G(C, T)$ is defined to be the expectation of the outcome, $E(C, T)$.

An example of a solution concept expressed using these variables is:

$$\{C \in \mathcal{C} | \forall_{C' \in \mathcal{C}} \forall_{T \in \mathcal{T}} : \ G(C, T) \geq G(C', T)\}.$$

This solution concept is known as 'simultaneous maximization of all outcomes'. As it requires that there is a single solution that maximizes the outcomes against all possible tests, this is a very strong strong solution concept, but has limited application scope. In [15] an brief overview of various other solution concepts is given, among which the Nash-equilibrium, which we will treat in the next section.

7.1.2 Memory

An often encountered problem in coevolutionary approaches is that of *forgetting* [21], i.e., certain components of behavior, or *traits*, are lost in a next generation only to be needed again at a later stage. This is especially the case for games with intransitive cycles, such as the Rock-Scissors-Paper game, discussed in section 4.2.2.

In order to counter this forgetting of trades, *memory mechanisms* are employed. The idea is that in the coevolutionary path to the solution concepts various traits will have to be discovered. Traits that constitute the solution will have to be remembered by the memory.

7.2 Nash equilibrium solution concept

In this section we give an outline of a memory mechanism for reaching the Nash-equilibrium solution concept for symmetric zero-sum games as presented in [21] ("Nash-memory").

7.2.1 Symmetric games and Nash equilibria

In a symmetric game the form of the policy for both players is identical: they can take the same actions in the same information sets [1], as is the case in Rock-Scissors-Paper. Put differently: both players select their (possibly mixed) policy from the same set of pure policies available for the game.

Symmetric zero-sum game always have a value 0, because this is the expectation of a policy played against itself: $\forall_\pi \, E(\pi, \pi) = 0$ or, expressed in terms of

[1] This implies players take actions simultaneous.

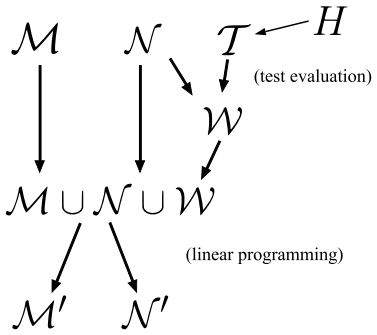


Figure 7.1: One iteration of Nash-memory coevolution.

candidate solutions and tests: $\forall_\pi G(\pi, \pi) = 0$. This means that a Nash equilibrium policy provides a security-level payoff of 0 and that therefore we are searching for a, usually mixed, policy π such that $\forall_{\pi'} G(\pi, \pi') \geq 0$.

Let $S(\pi)$ denote the *security set* of policy π, i.e., $S(\pi) = \{\pi' | G(\pi, \pi') \geq 0\}$. Now, the Nash-equilibrium solution concept can be expressed as:

$$\{\pi | \forall_{\pi'} : \pi' \in S(\pi)\}.$$

7.2.2 Components of the Nash-memory

Let $\mathcal{N}$ and $\mathcal{M}$ be two mutually exclusive sets of pure policies. $\mathcal{N}$ is defined to be the support of mixed policy $\pi_\mathcal{N}$ which will be the approximation of the Nash-policy during the coevolution process. Therefore this is the candidate solution.[2]

The policies that are not in $\mathcal{N}$ are not needed by $\pi_\mathcal{N}$ to be secure against all encountered policies. These unused policies are stored in the set $\mathcal{M}$. The fact that $\pi_\mathcal{N}$ is secure against all policies means that $\mathcal{N} \cup \mathcal{M} \subseteq S(\pi_\mathcal{N})$. Put in coevolutionary terms, $\mathcal{M}$ holds those policies, that are currently not needed to be secure against all encountered policies ($\mathcal{N} \cup \mathcal{M}$), in order not to forget particular traits they might embody.

Apart from the candidate solution $\pi_\mathcal{N}$ and an additional memory $\mathcal{M}$, the Nash-memory mechanism specifies a search heuristic H. This is an arbitrary heuristic that delivers new tests against which the candidate solution is evaluated.

7.2.3 The operation

We now turn to the actual working of the Nash-memory. To start, $\mathcal{M}$ is initialized as the empty set and $\mathcal{N}$ is initialized as a set containing an arbitrary pure policy and $\pi_\mathcal{N}$ as the 'mixed' policy that assigns probability 1 to this pure policy.[3] Then the first iteration begins.

[2] An alternative view is that the Nash-memory maintains a 'population' of candidate solutions consisting of *one* individual, which in turn consists of multiple of pure policies.

[3] In [21] the initialization is taken somewhat different, but this doesn't affect the working of the memory mechanism.

Algorithm 3 Nash-memory mechanism

```
πN= initializePolicy
N = support(πN)
M = ∅
For iteration = 1:nr_iterations
  W = ∅
  T = H()    //set of tests from search heuristic
  Forall t in T
    If G(t,πN) > 0
      W = W ∪ {t}
    End
  End
  all_policies = N ∪ M ∪ W
  // Calculate a new policy secure against all_policies with
  // linear programming:
  πN = LP(all_policies)
  N = support(πN)
  M = all_policies \ N   // unused policies stored in M
End
```

Figure 7.1 shows one iteration of the Nash-memory. First, a set of test-policies, $\mathcal{T}$, is delivered by the search heuristic. The policies in this set are evaluated against $\pi_{\mathcal{N}}$, to define the set of 'winners':

$$\mathcal{W} = \{\pi \in \mathcal{T} | G(\pi, \pi_{\mathcal{N}}) > 0\}.$$

When this set is non-empty, clearly $\pi_{\mathcal{N}}$ is not a Nash-equilibrium policy, as it is not secure against all policies, and therefore should be updated.

First a payoff matrix of all policies in $\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$ played against each other is constructed.[4]In this matrix the rows correspond to policies played by the first player, the columns to those of the second player. The entry (i, j) gives the (expected) outcome of policy i against j, $G(\pi_i, \pi_j)$.

This matrix can than be used to define a linear program. Relating to section 2.2.3 and 2.3.4. the payoff matrix corresponds with $\mathbf{A}$. Therefore this can be solved as outlined in section 2.3.4. The result will be the new policy $\pi'_{\mathcal{N}}$, the policies to which it assigns positive weight is the new set $\mathcal{N}'$, the other policies are stored in $\mathcal{M}'$.

The full algorithm is shown on the current page. Because $S(\pi_{\mathcal{N}})$, the set of pure policies against which $\pi_{\mathcal{N}}$ is secure, grows monotonically with each iteration, repeated application will converge to a Nash-equilibrium, provided that the search heuristic is able to find policies that beat our current estimate (that is, a non-empty $\mathcal{W}$ is found).

When resources are limited, it might not be feasible to store all policies encountered. Therefore it is possible to limit the size of $\mathcal{M}$, by discarding policies that have not been recently used by $\pi_{\mathcal{N}}$ using some heuristic. This, however, might re-introduce the problem of forgetting and will therefore not be considered any further in this thesis.

[4]Of course the outcomes of pure policies in $\mathcal{M} \cup \mathcal{N}$ against each other can be cached, so only the outcomes of policies from $\mathcal{W}$ against other policies will have to be calculated to construct this matrix.

7.3 Coevolution for 8-card poker

In this section we apply the Nash-memory mechanism on 8-card poker. In doing so, we extend the Nash-memory for usage with asymmetric games.[5] Secondly, we use the method to calculate a best-response policy as described in chapter 3 to generate new tests. I.e., the search heuristic H we use is a procedure $bestResponse(\pi)$ that constructs and solves a POMDP model of the game played against an opponent that uses policy π.

7.3.1 Asymmetric games

In order to apply the Nash-memory mechanism to 8-card poker, we need an extension to allow tackling asymmetric games.

A simple solution is to create a new compound game consisting of two games of 8-card poker; one played as gambler and one played as dealer. This compound game is symmetric and a particular policy i is given by $\pi^i = \left\langle \pi^i_{gambler}, \pi^i_{dealer} \right\rangle$. We refer to this as *naive symmetrization*.

Using this new representation the Nash-memory mechanism can directly be applied without further changes. However, it is clear that the flexibility with which the new mixed policy is constructed is constrained: it is not possible to put more weight on a particular gambler policy $\pi^i_{gambler}$ without putting the same weight on the corresponding dealer policy π^i_{dealer}.

In order to overcome this limitation we propose an extension of naive symmetrization. Observe that in algorithm 3 there are only two reasons why the game must be symmetric: to determine whether a test policy beats the current mixed policy, $G(t, \pi_{\mathcal{N}}) > 0$, and because the next Nash-approximation is constructed from all encountered policies ($\mathcal{M} \cup \mathcal{N} \cup \mathcal{W}$).

To overcome this, the proposed symmetrization applies the Nash-memory mechanism per player. I.e,. we maintain one sets $\mathcal{N}_p, \mathcal{M}_p, \mathcal{W}_p, \mathcal{T}_p$ and a Nash-approximation, $\pi_{p,\mathcal{N}}$, for each player $p = 1, 2$ (gambler, dealer). If, without loss of generality, we assume that the search heuristic delivers a single test policy for both players, T_1 and T_2, we can test whether the compound policy $T = \langle T_2, T_1 \rangle$[6] beats the compound policy $\pi_{\mathcal{N}} = \langle \pi_{1,\mathcal{N}}, \pi_{2,\mathcal{N}} \rangle$, as:

$$G(T, \pi_{\mathcal{N}}) = G(T_2, \pi_{2,\mathcal{N}}) + G(T_1, \pi_{1,\mathcal{N}}).$$

If $G(T, \pi_{\mathcal{N}}) > 0$, then the current Nash-approximation, $\pi_{\mathcal{N}}$, is not secure against compound policy T. In this case the components of T are taken to be 'winners': $W_1 = T_2$ and $W_2 = T_1$.[7]

This results in two sets $\mathcal{M}_1 \cup \mathcal{N}_1 \cup \mathcal{W}_1$ and $\mathcal{M}_2 \cup \mathcal{N}_2 \cup \mathcal{W}_2$ with pure policies for respectively gambler and dealer. By constructing the payoff matrix for these pure policies and applying linear programming we calculate $\pi'_{1,\mathcal{N}}$ and $\pi'_{2,\mathcal{N}}$, from which $\mathcal{M}'_1, \mathcal{N}'_1, \mathcal{M}'_2$ and $\mathcal{N}'_2$ are constructed. The compound policy:

$$\pi'_{\mathcal{N}} = \left\langle \pi'_{1,\mathcal{N}}, \pi'_{2,\mathcal{N}} \right\rangle,$$

[5] It is already indicated in [21] that such an extension is possible.

[6] Note that a test policy T_1 for player 1, is a policy for his opponent, player 2, and vice versa.

[7] The remark from note 6 applies here too.

Algorithm 4 Asymmetric Nash-memory using *bestResponse* heuristic.

```
For p = 1,2          //for both players
  π_{p,N} = initializePolicy(p)
  N(p) = support(π_{p,N})
  M(p) = ∅
End
While !converged
  For p = 1,2
    N_stoch(p) = mixed2StochasticPolicy(π_{p,N}))
    T(p) = bestResponse(N_stoch(p))
  End
  G(T,π_N) = G(T(1), π_{1,N}) + G(T(2), π_{2,N}))
  If G(T,π_N) > 0
    For p = 1,2
      W(modulo(p,2)+1) = T(p)
      NMW(p) = N(p) ∪ M(p) ∪ W(p)
    End
    π_{1,N}, π_{2,N} = LP(NMW(1),NMW(2))
    For p = 1,2
      N(p) = support(π_{p,N})
      M(p) = NMW(p) \ N(p)
    End
  Else
    converged = true;
  End
End
```

is secure against all combinations of gambler and dealer policies from $\mathcal{M}'_1, \mathcal{N}'_1, \mathcal{M}'_2$ and $\mathcal{N}'_2$ in the compound game.

7.3.2 Best-response heuristic

The search heuristic is an important aspect for coevolutionary approaches. It should be powerful enough to discover improvements to the current candidate solution. Within the Nash-memory mechanism this means that it has to find policies that beat the current Nash-approximation.

The approach as outlined in chapter 3 provides a suitable candidate: calculating the best-response policies against the current Nash approximations, $\pi_{1,\mathcal{N}}, \pi_{2,\mathcal{N}}$. The best-response policies obtain the highest payoff possible. A desirable side effect is that this provides a convergence criterion: when the best-response policies are not able to attain a positive payoff in the compound game, i.e. $G(T, \pi_{\mathcal{N}}) = 0$, then $\pi_{\mathcal{N}}$ is a Nash-policy.

However, using the approach from chapter 3 we can calculate a best response against a stochastic policy. In contrast, the Nash-approximations, are mixed policies. This means it is necessary to convert a mixed policy to a stochastic policy. For now we assume this is done by a procedure *mixed2StochasticPolicy*. How this procedure works will be covered in detail in section 7.4.

policy		information sets					
number	probability	J	Q	K	Jb	Qb	Kb
1	.2	1	1	1	1	1	1
2	.3	1	0	1	0	1	1
3	.5	0	0	1	0	0	1

Table 7.1: A mixed policy for the gambler in 3-card poker. Shown is the probability of betting ('1').

7.3.3 The resulting algorithm

The resulting algorithm is shown on the preceding page. Note that $\mathcal{M}_p, \mathcal{N}_p, T_p, \mathcal{W}_p$ are denoted $M(p), N(p), T(p), W(p)$ with $p = 1, 2$ representing the player number.

The expression $modulo(p, 2) + 1$ assures that the assignments $W_1 = T_2$ and $W_2 = T_1$ are performed as explained in section 7.3.1.

The procedure $LP()$ constructs the payoff matrix for the two sets of policies and solves the linear program defined. The entries in the payoff matrix can be cached to prevent re-calculating outcomes between pairs of pure policies. In particular, because only one pair of new policies is provided per iteration, only the outcomes of these have to be evaluated against the policies already in memory, i.e. W_1 against $\mathcal{M}_2, \mathcal{N}_2, \mathcal{W}_2$ and vice versa.

7.4 From mixed to stochastic policies

In this section we explain how we can transform a mixed policy to a equivalent stochastic policy. First we will re-introduce some relevant concepts and illustrate the problem. Next, in section 7.4.2 we show that the realization weights are an adequate tool to tackle this problem and after that we discuss computing them.

7.4.1 Problem and concepts

Recall a policy is a mapping from information sets to actions. A deterministic or pure policy specifies exactly one action for each information set. A stochastic policy, on the other hand, is a single policy that specifies a probability distribution over actions for each information set.

A mixed policy is a set of, usually pure, policies together with a probability distribution over this set.[8] Intuitively it is possible, at least for tree-like games, to convert a mixed policy to a stochastic policy. Exactly how to do this is not trivial, though.

We will make use of an example 3-card poker game. It is exactly like 8-card poker only with three cards: J, Q and K. Table 7.1 shows a mixed policy for the gambler for this game. Shown are the information sets the gambler has in this game and the probability of betting in those information sets according to 3 policies. Also shown are the probabilities of playing each of the three policies.

A naive approach to convert the mixed policy shown would be to multiply to the rows, i.e the probabilities of betting according to the policies, with the

[8] In general, the policies in the set can also be stochastic, but not mixed, policies themselves.

probability of the respective policy and add the results. This problem with this approach, however, is that does not respect the fact that the chance of reaching an information set also depends on the policy. Expressed differently, it does not take into concern the probability that a policy *realizes* a certain move.

Example 7.4.1 As an example consider information set 'Jb' in table 7.1. When applying the naive approach the probability of betting in the resulting stochastic policy would become $0.2 \cdot 1 + 0.3 \cdot 0 + 0.5 \cdot 0 = 0.2$. In the original mixed policy, however, policy number 1 would specify 'bet' ('1') after observing the jack (information set 'J'). Therefore information set 'Jb' would never be realized using policy 1. As the other policies specify never to bet at 'Jb', the probability of betting at 'Jb' in the stochastic policy is therefore 0. □

In the above the word 'realizes' is stressed with good reason. The problem in concern is very much related to the sequence form and its realization weights.

Recall from section 2.3.2 that a sequence corresponds with a path from the root of the game-tree to a node n. The sequence $\sigma_k(n)$ for player k is the string consisting of the concatenation of all labels of edges corresponding with player k's moves and observations. Equivalently, a sequence $\sigma_k(n)$ corresponds with an information set of player k concatenated with an action that can be taken at that information set. As each node from the tree corresponds to exactly one sequence, the number of sequences, m, is bounded. We also write σ_k^l, with $1 \leq l \leq m$.

Also recall that the realization weight $\rho_k^i(\sigma_k^l)$[9] of a sequence σ_k^l under policy π_k^i for player k, is defined as a conditional probability: $\rho_k^i(\sigma_k^l)$ is the probability that player k takes all the actions specified by σ_k^l *given that all corresponding information sets are reached.*

In the next subsection, we will show that the realization weights are an appropriate tool for our problem of converting a mixed policy μ_k for player k to a stochastic one.

7.4.2 Using realization weights

Here we show that using realization weights, we can transform a mixed policy to a stochastic policy that describes the same dynamics, i.e, induces the same outcomes.

Formally, we want to find the probability of an action a at an information set I_k, $P(a|I_k, \mu_k)$ corresponding to μ_k for player k. The crucial step in this problem is that we have to weight the contribution to $P(a|I_k, \mu_k)$ of a policy $\pi_k^i \in \mu_k$ by the probability that information set I_k is actually realized by π_k^i.

Theorem 7.4.1 *To transform a mixed policy μ_k for player k to a stochastic policy, realization weights for all policies $\pi_k^i \in \mu_k$ are sufficient. For a particular action a and information set I_k, the stochastic policy is given by:*

$$P(a \mid I_k, \mu_k) = \frac{\sum_i P(\pi_k^i) \cdot \rho_k^i(\sigma_k(I_k'))}{\sum_i P(\pi_k^i) \cdot \rho_k^i(\sigma_k(I_k))}, \tag{7.1}$$

where $\sigma_k(I_k)$ is the sequence that leads to information set I_k and $\sigma_k(I_k')$ is the sequence that result from appending action a to $\sigma_k(I_k)$.

[9] We denote the realization weight with ρ here.

Proof When N players play policies $(\pi_1, ..., \pi_N)$, the probability of reaching a node n in the game-tree is given by:

$$P(n|\pi_1, ..., \pi_N) = \beta(n) \cdot \prod_{k=1}^{N} \rho_k(\sigma_k(n)),$$

where $\beta(n)$ is the product of all chance moves on the path from the root to n. In order to discriminate between the probabilities of moves of the player in concern and its opponents, this can be rewritten to:

$$P(n|\pi_1, ..., \pi_N) = \rho_1(\sigma_1(n)) \cdot \beta(n) \cdot \prod_{k=2}^{N} \rho_k(\sigma_k(n)),$$

in which $k = 1$ is an arbitrarily chosen player we focus on. Similarly, the actual chance of reaching a particular information set I_1 can be given as:

$$P(I_1|\pi_1, ..., \pi_N) = \sum_{n \in I_1} \left(\rho_1(\sigma_1(n)) \cdot \beta(n) \cdot \prod_{k=2}^{N} \rho_k(\sigma_k(n)) \right).$$

As player 1 can not discriminate between the nodes of I_1, clearly his sequences for these nodes are the same and we write $\sigma_1(I_1)$, giving:

$$P(I_1|\pi_1, ..., \pi_N) = \rho_1(\sigma_1(I_1)) \cdot \sum_{n \in I_1} \left(\beta(n) \cdot \prod_{k=2}^{N} \rho_k(\sigma_k(n)) \right).$$

Now let $P_{opp} = \sum_{n \in I^j} \left(\beta(n) \cdot \prod_{k=2}^{N} \rho_k(\sigma_k(n)) \right)$ denote the opponent (and nature) component of the realizing I_1. When there are multiple policies $\pi_1^i \in \mu_1$, each played with a probability of $P(\pi_k^i)$, the probability of realizing I_1 becomes:

$$P(I_1|\mu_1, P_{opp}) = P_{opp} \cdot \sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1)).$$

Next we turn our attention to realizing both I_1 and the desired action a. For a single policy $\pi_1^i \in \mu_1$, this probability is:

$$P(I_1, a|\pi_1^i, P_{opp}) = P_{opp} \cdot \rho_1^i(\sigma_1(I_1)) \cdot P(a|\pi_1^i, I_1).$$

For the mixed policy μ_1 this becomes:

$$P(I_1, a|\mu_1, P_{opp}) = P_{opp} \cdot \sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1)) \cdot P(a|\pi_1^i, I_1).$$

Finally we can give the probability of action a *given* I_1 for mixed policy μ_1:

$$\begin{aligned} P(a \mid I_1, \mu_1, P_{opp}) &= \frac{P(I_1, a|\mu_1, P_{opp})}{P(I_1|\mu_1, P_{opp})} \\ &= \frac{P_{opp} \cdot \sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1)) \cdot P(a|\pi_1^i, I_1)}{P_{opp} \cdot \sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1))} \\ &= \frac{\sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1)) \cdot P(a|\pi_1^i, I_1)}{\sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1))} \\ &= P(a \mid I_1, \mu_1). \end{aligned} \tag{7.2}$$

Now note that the sequence $\sigma_1(I_1)$ followed by action a defines a new sequence, let's call this sequence $\sigma_1(I'_1)$. The realization weight of this new sequence under policy i is $\rho_1^i(I'_1) = \rho_1^i(\sigma_1(I_1)) \cdot P(a|\pi_1^i, I_1)$. Therefore we can rewrite equation 7.2 totally in terms of priors and realization weights:

$$P(a \mid I_1, \mu_1) = \frac{\sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I'_1))}{\sum_i P(\pi_1^i) \cdot \rho_1^i(\sigma_1(I_1))}.$$

Now observing that the focus on player $k = 1$ was an arbitrary choice and that this procedure can be extended for any information set and action proves the theorem. □

7.4.3 Calculating realization weights

Having established that realization weights for the policies $\pi_k^i \in \mu_k$ will give the solution to the problem, the next goal is to determine them. In contrast to the Gala system, we do not want to find realization weights that define an optimal policy, but simply want to extract the realization weights from a policy π_k^i.

Let σ_k be the sequence for reaching some node n where player k is to move. Then the continuation $\sigma_k \circ a$ is also a sequence for player k and the realization weights are given by the following recurrence relation:

$$\rho_k^i(\sigma_k \circ a) = \rho_k^i(\sigma_k) \cdot P(a|\pi_k^i, n). \tag{7.3}$$

Because $P(a|\pi_k^i, n)$ is a probability distribution that sums to 1 [10], the total realization weight of continuations of a sequence, σ_k, sum to the realization of that sequence itself. I.e $\rho_k^i(\sigma_k) = \rho_k^i(\sigma_k \circ a_1) + ... + \rho_k^i(\sigma_k \circ a_n)$, exactly as was required in section 2.3.3.

As $\rho_k^i(root) = 1$ for any policy i, starting at the root and iteratively applying equation 7.3 while walking through the game-tree extracts all the realization weights.

We can also formulate this slightly different. Recall that in the proof of theorem 7.4.1, we wrote $\sigma_k(I_k)$ for the sequence for player k for reaching any node in I_k, an information set for that player. When using this notation for equation 7.3, we get:

$$\rho_k^i(\sigma_k(I_k) \circ a) = \rho_k^i(\sigma_k(I_k)) \cdot P(a|\pi_k^i, I_k).$$

Now, observe that the continuation $\sigma_k(I_k) \circ a$ will correspond with the sequence for all successor information sets, I'_k, that can be reached from I_k when action a is chosen. By formalizing this it is possible to express everything in terms of information sets.

Definition 7.4.1 The realization weight of an information set I_k of player k under a policy π_k^i will be denoted $\rho_k^i(I_k)$ and is defined as the realization weight of the sequence of any node $n \in I_k$:

$$\rho_k^i(I_k) :\equiv \rho_k^i(\sigma_k(n)).$$

Note, that the realization weight of an information set of another player, i.e., $\rho_k(I_l),\ k \neq l$ is undefined.

[10] In this setting where we considered pure policies π_k^i, $P(a|\pi_k^i, n)$ is 1 for exactly one action. In general, however, a mixed policy might also have stochastic policies in its support.

Algorithm 5 Calculate information set realization weights(π_k)

```
Forall IS in initial_ISs(k)
  rw(IS)=1
  append(ISq,IS) //ISq is a queue
End
While !empty(ISq)
  IS=pop(ISq)
  Forall a in ACTIONS
    Forall sucIS in successor_ISs(IS,a,k)
      rw(sucIS)=rw(IS)·P(a|IS,π_k)
      append(ISq,sucIS)
    End
  End
End
```

As above, let I'_k be any information set for player k, that can be reached from I_k when playing a. The recurrence relation now becomes:

$$\rho^i_k(I'_k) = \rho^i_k(I_k) \cdot P(a|\pi^i_k, I_k). \tag{7.4}$$

This formulation expresses the close relation between information sets, action probabilities and realization weights more naturally. Also it gives a further formalization of the step taken to obtain equation 7.1 from equation 7.2. Using definition 7.4.1, the latter can be rewritten as:

$$P(a \mid \mu_k, I_k) \quad = \quad \frac{\sum_i P(\pi^i_k) \cdot \rho^i_k(I_k) \cdot P(a|\pi^i_k, I_k)}{\sum_i P(\pi^i_k) \cdot \rho^i_k(I_k)}, \tag{7.5}$$

consecutively applying 7.4 gives:

$$P(a \mid \mu_k, I_k) \quad = \quad \frac{\sum_i P(\pi^i_k) \cdot \rho^i_k(I'_k)}{\sum_i P(\pi^i_k) \cdot \rho^i_k(I_k)}.$$

Backwards substitution using definition definition 7.4.1, then immediately gives equation 7.1.

The new recurrence relation (eq. 7.4) also defines an algorithm to find the realization weights for information sets very naturally. This algorithm is shown on the current page and consists of two phases: the first phase finds all initial information sets for player k, that are the information sets in which the player makes his first move of the game. The realization weights of these information sets are initialized to 1.[11] The second phase consists of a pass through the game-tree finding successor information sets and calculating their realization weights.

[11] The sequence of an initial information set, is the root sequence, $\emptyset$.

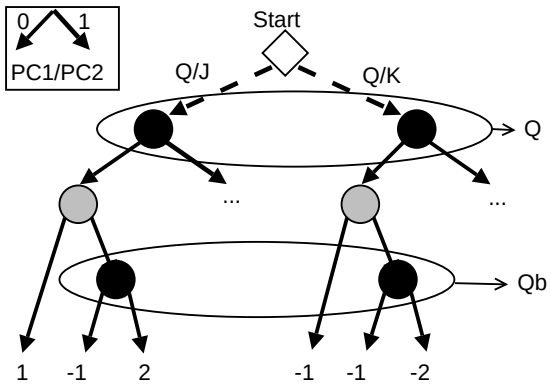


Figure 7.2: Partial game-tree for 3-card poker. The information sets Q and Qb for the first player are clearly indicated.

7.4.4 Calculating the stochastic policy

At this point, calculating a stochastic policy from a mixed policy has become almost trivial. Once the realization weights for the information sets are calculates, all one has to do is apply equation 7.5. We will give an example for the mixed policy from table 7.1.

Example 7.4.2 Figure 7.2 shows a part of the game-tree for 3-card poker. It shows 2 information sets: Q and Qb. In this example we will calculate the stochastic policy for these information sets.

The first thing we need to do is calculating the realization weights of the information sets under the different policies that make up the mixed policy from table 7.1.

As the gambler makes its first move when in Q, this is an initial information set and therefore its realization weight is 1 under all policies. In contrast Qb is not an initial information set and its realization weight is given by:

$$\rho^i(Qb) = \rho^i(Q) \cdot P(\text{`0'}|\pi^i, Q),$$

where '0' indicates the action pass.[12] This leads to the following table of realization weights:

policy	$\rho^i(Q)$	$\rho^i(Qb)$
1	1	0
2	1	1
3	1	1

Table 7.2: Realization weight for the policies in the support of the mixed policy.

Now we can apply fill out equation 7.5 for Q, yielding:

[12]Note we omit the subscripts indicating the player (which is 'gambler' throughout this whole example).

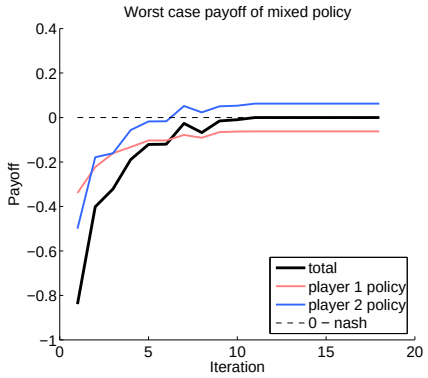


Figure 7.3: Results for the Nash-memory approach to 8-card poker. The dashed lines indicate the Nash-value.

$$\begin{aligned} P(\text{`1'} \mid \mu, Q) &= \frac{\sum_i P(\pi^i) \cdot \rho^i(Q) \cdot P(\text{`1'}|\pi^i, Q)}{\sum_i P(\pi^i) \cdot \rho^i(Q)} \\ &= \frac{0.2 \cdot 1 \cdot 1 + 0.3 \cdot 1 \cdot 0 + 0.5 \cdot 1 \cdot 0}{0.2 \cdot 1 + 0.3 \cdot 1 + 0.5 \cdot 1} \\ &= \frac{0.2}{1} = 0.2. \end{aligned}$$

For Qb this gives:

$$\begin{aligned} P(\text{`1'} \mid \mu, Qb) &= \frac{\sum_i P(\pi^i) \cdot \rho^i(Qb) \cdot P(\text{`1'}|\pi^i, Qb)}{\sum_i P(\pi^i) \cdot \rho^i(Qb)} \\ &= \frac{0.2 \cdot 0 \cdot 1 + 0.3 \cdot 1 \cdot 1 + 0.5 \cdot 1 \cdot 0}{0.2 \cdot 0 + 0.3 \cdot 1 + 0.5 \cdot 1} \\ &= \frac{0.3}{0.8} = 0.375. \end{aligned}$$

Concluding the example. □

7.5 Experiments

In this section we will describe some experiments performed using the Nash-memory mechanism as outlined in this chapter.

7.5.1 8-card poker

Algorithm 4 was implemented and applied to 8-card poker. Figure 7.3 shows the obtained results. It shows that it only takes a few iterations to obtain a policy that is fairly secure. This is a nice property, as it indicates that this technique might be applied for larger games to obtain an approximate Nash policy.

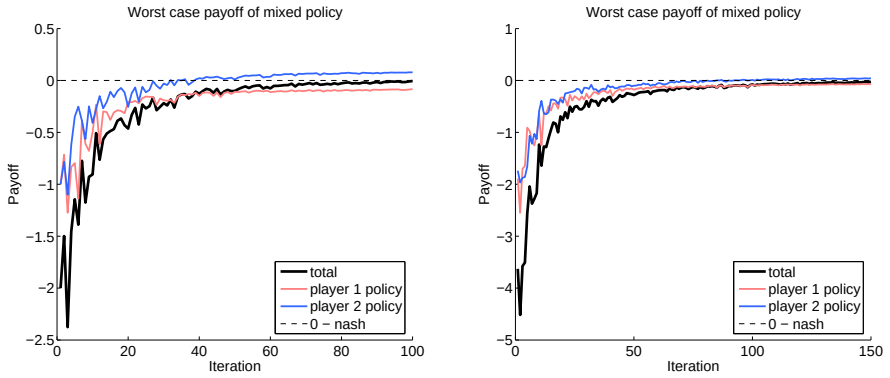


Figure 7.4: Two larger poker-games. Left: 4-bet 8-card poker. Right: 2-round, 3-bet-per-round 6-card poker.

It also indicates that only a relatively small number of policies is needed to be secure. Further investigation made this even more plausible, as it turned out that the number of pure policies used by the mixed policy is even lower than the figure suggests: when reaching the Nash level (iteration 12) only 6 out of 12 pure policies are assigned weight for the both gambler and dealer policy.

Another observation that can be drawn from the figure is that, although convergence to Nash equilibrium is monotonic, because with each iteration the approximate Nash becomes secure against more policies[13], the worst case payoff does not increase monotonically. Apparently, a particular policy against which it is not secure yet might become a best-response and do more damage.

7.5.2 Some larger poker games

After the encouraging results for 8-card poker some experiments were performed on larger poker games. We show resulting curves for two of them here. The first is an 8-card poker variant that allows up betting u to 4 coins bets, with a maximum raise of 2 coins. The game-tree for this game contains nearly 4000 nodes and has 274 sequences for each player.

The second game is a 2 round poker game with a deck of 6 cards, both players receive one card and play a bet-round, after which 1 public card appears face-up on the table. Then a final bet-round is played. In both bet-rounds a maximum of 3 coins coins can be betted per player. This game-tree for this game consists of over 18,000 nodes and has 2162 sequences for both players.

For these games, the obtained results are shown in figure 7.4. As was the case for 8-card poker, the Nash-memory is able to obtain a reasonable security level in a relatively low number of iterations.

Also the small number of policies needed for the support of the mixed policy was confirmed for these larger games. For 4-bet 8-card poker $\mathcal{N}$ contained 18 policies out of 100 on convergence. At iteration 150 for the 6-card poker game, the number of policies with positive weight was 29.[14]

[13] More formal, the set $S(\pi_{\mathcal{N}})$ grows monotonically.

[14] The algorithm was not fully converged at this point, as the compound policy still received

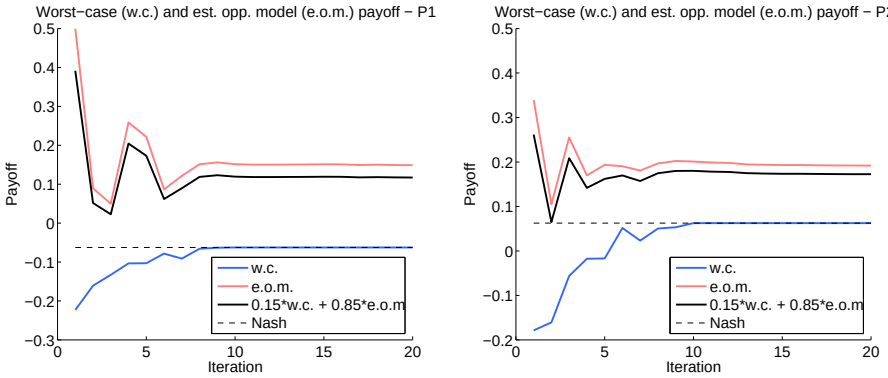


Figure 7.5: The tradeoff between security and higher payoff for 8-card poker. The estimated opponent model is uniform random.

For the larger games there seem to be more oscillations in worst-case payoff. This can probably be explained in the following way: because the game-tree for these games is larger and the horizon is deeper, more actions affect later stages of the game. Therefore the relatively small adjustment of the mixed policy can influence the realization weights of a lot of information sets. When a particular set of information sets is given more weight, but the policy specified for this set is not optimal, this can be exploited by the opponent.

7.5.3 Security vs. best-response payoff

As argued before, playing a Nash-equilibrium is too conservative, when the opponent is not expected to play optimal. On the other hand playing a best-response policy may present risks, as the opponent model may be inaccurate. In this experiment a way to find a tradeoff between potential winnings and security is examined.

The idea is as follows. The opponent model delivers two estimated opponent policies, one gambler and one dealer policy.[15] First, the best-response policies against these estimated opponent policies are calculated. These best-response policies are used to initialize the Nash-memory mechanism, which then is run until convergence. The result is a series of mixed policies (for both gambler and dealer), starting with the best-response against the estimated opponent policy and ending with a Nash-equilibrium.

Each of these resulting mixed policies, however, can also be evaluated against the estimated opponent policy. When we do this for all of them, we know the worst-case payoff and the outcome against the estimated opponent model.

Figure 7.5 shows this evaluation for 8-card poker. It also shows a line that is a weighted average between the worst-case payoff and that obtained against the estimated opponent model. One should interpret the weights for this line (0.85 : 0.15 in this case) as the amount of trust in the opponent model versus

a worst case payoff of -0.027 instead of 0.

[15] Expressed differently, it delivers an estimated opponent policy for the compound game.

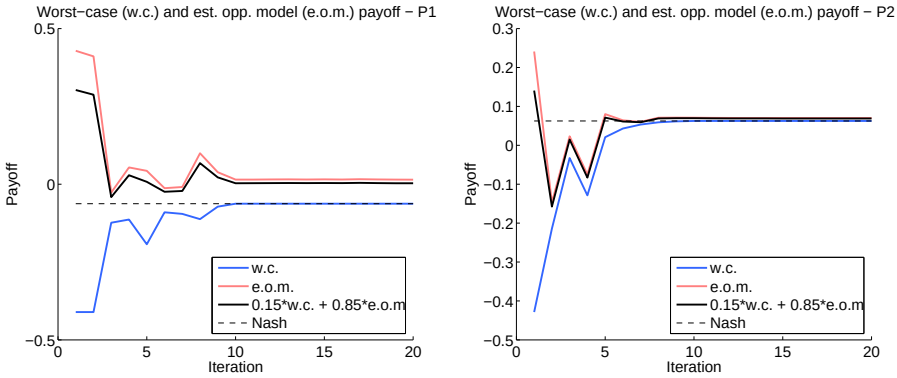


Figure 7.6: The security / potential winnings tradeoff for another estimated opponent. Especially for player 2 there are no useful peak values between the best-response and Nash-policy.

the amount of trust that the opponent plays a best-response against the mixed policy.

Given such a trust ratio, any existing peeks in the weighted average identify those mixed policies that have a beneficial estimated- and worst-case outcome with respect to the amount of trust. As a consequence these policies should be considered a candidate. We have not considered which of these mixed policies should actually be used. One idea would be to randomly choose between them.

Unfortunately, whether these peek policies exist depends very much on the estimated opponent model. An example in which these peeks are missing is shown in figure 7.6. In particular the procedure seems to fail to identify useful mixed policies, when the best-response (or some other 'good'-response) against the estimated opponent model is not in the support of a Nash equilibrium.

Another issue observed is that the payoff against the estimated opponent is much larger for the first (best-response) policy than for any of the mixed policies.

7.6 Discussion

When comparing the Nash-memory approach with solving the sequence form (as in Gala) with respect to performance there are a couple of interesting differences. At this point, calculating a Nash-equilibrium using the Nash-memory approach consumes more time. However, it spends its time differently: mostly on constructing and solving the POMDP models, to calculate the best response, and determining outcomes between the encountered pure policies. Far less time is spent on linear programming, as the size of the linear programs to be solved is generally smaller. E.g. for the 2-round 6-card poker experiment the maximum size of the matrix was 150×150 versus 2162×2162 for solving sequence form. Also, the linear programs solved have a simpler constraint matrix (a row matrix, forcing the weights of the pure policies to sum to 1).

We expect that considerable speed-up can be obtained by streamlining the of

implementation of POMDP model construction and solving. Moreover, approximate methods could be used for both solving the POMDP and evaluating the rewards. This might lead to this approach becoming competitive the sequence form solving in terms of performance. The anytime nature of the Nash-memory approach makes it even more appropriate for a lot of domains.

We will make a few remarks regarding the tradeoff as explained in section 7.5.3. Perhaps the best-response heuristic is not the most appropriate to use during the operation of the Nash-memory with as goal to search for a suitable candidate policy that trades off potential gain for security. There is a large gap between a failing opponent model and the opponent predicting our policy acting to minimize our profit. Put differently, perhaps the worst-case payoff is a too negative measure and we need to search for a weaker form of security.

A direction for this could be to analyze the type and magnitude of errors made by an opponent model. When this knowledge is available it could be possible to generate other opponent policies that fall within the expected bounds of error for the opponent model. The Nash-memory mechanism can than be employed to construct policies that are secure against all of them.

A different question regarding the Nash-memory mechanism that needs more research is the following. Currently the Nash memory is based on mixed policies. Would it be possible to directly use stochastic policies, or policies expressed in terms of realization weights? In this case we would not need to convert between mixed and stochastic policies as explained in section 7.4.

Another direction of future research would be to try to avoid solving a linear programming from the start in each iteration. There might be an approach to adjust the weights of the mixed policy without solving a complete linear program.

A final pointer is to focus on extending this approach to games with multiple players or games that are not zero-sum. A form of symmetrization might also be possible in this case. Calculating a best-response against two (or more) fixed opponents can be done by transforming the game to a POMDP, finding a secure mixture of policies could be done using any of the methods described in [45]. Perhaps an incremental weight-adjusting algorithm, as mentioned above, will also provide opportunities for these directions.

Chapter 8

Conclusions

In this thesis we have addressed partially observable card games, specifically poker games. In our covering of these games, we have shown two perspectives: the game theoretic approach, that specifies a Nash-equilibrium that guarantees a security level payoff and an agent centric (POMDP) approach, yielding a best-response policy that exploits weaknesses of a given opponent policy. We have experimentally shown that the POMDP approach was able to obtain the maximum payoff, even against a Nash-policy.

Next, we presented an investigation of methods that allow for tackling large POMDPs and thus larger poker games. In particular we discussed model minimization for POMDPs and made plausible that direct application of this method will not give enough reduction for real-life poker variants as Texas' Hold-em. We also identified the bottleneck and gave a pointer to a potential solution.

Finally, we considered an alternative way of calculating Nash-equilibria using a coevolutionary approach. This process also gives a natural way to identify policies that make a beneficial tradeoff between security and potential gain. Although it depends on the opponent policy and the used search heuristic whether a policy giving a favorable tradeoff is found. This can be seen as a first step in unifying the game theoretic and agent centric approach.

8.1 Future work

Most directions for future work were identified in the last two parts of this thesis. As mentioned above, in the second part a modification for model minimization for POMDPs is suggested. Future research should focus on whether this modification still allows for a equivalence notion that satisfies the original bisimulation theorem (5.2.1). If this is possible, it would be interesting to see whether such a new aggregation concept will alow for tackling real-life poker games.

Apart from state aggregation such as model minimization, we also briefly discussed other approaches for dealing with large (PO)MDPs. The most relevant leads that were identified are the approximate methods. Especially the trajectory sampling approaches seem promising, as they provide performance bounds independent of the number of states.

Roughly speaking, we identified three types of future work in the last part.

The first type would be to try and generalize the coevolutionary computation of Nash equilibria to games with more than two players or games that are not zero-sum. A second type would be to try to prevent solving a linear program from start, by using some weight adjusting scheme. The last type would be to focus on the tradeoff between worst-case (security) and best-case (best-response) payoff. This direction would involve investigating different search heuristics that present opponent policies that are closer to the estimated opponent model.

A more general question that would be interesting for future research is whether the concept of realization weights can be generalized to arbitrary MDPs. As illustrated in this thesis, sequence form and their realization weights allow for more efficient operations in extensive form games. Therefore an extension of realization weights to arbitrary MDPs or POSGs might also present opportunities within these frameworks.

Appendix A

Gala system modifications

As mentioned in chapter 4, there were some changes necessary to get Gala up and running. Here we will briefly document these changes. Gala is written in SWI-prolog. Two of the changes were necessary to work with the current version (v. 5.4.3).

The Gala function *compare_ranks* (in poker.gl) needs to return '¡', '¿' or '=', because this is what the build in function *predsort* now requires as return arguments.

In the new versions of SWI-Prolog, operators are local to modules, therefore it is necessary to define the operators with the *user* scope.

Another necessary change involved the solving of the linear program. The Gala system included a Matlab file which used the deprecated lp function. This has been changed to use the 'linprog' function available in current releases. This new procedure takes its arguments in a different format. Also it was not clear whether the algorithm the new function implemented changed.

Except for the modification, also some practical additions have been made. These include a simulation module and various functions to extract policies and translate to understandable language and modify these policies.

Bibliography

[1] D. Billings. Computer poker. Master's thesis, University of Alberta, 1995.

[2] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proc. Int. Joint Conf. on Artificial Intelligence*, Acapulco, Mexico, August 2003.

[3] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artif. Intell.*, 134(1-2):201–240, 2002.

[4] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 493–499. American Association for Artificial Intelligence, 1998.

[5] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Poker as an experimental testbed for artificial intelligence research. In *Proceedings of AI'98, (Canadian Society for Computational Studies in Intelligence)*, 1998.

[6] K. Binmore. *Fun and Games*. D.C. Heath and Company, Lexington, MA, 1992.

[7] Craig Boutilier. Correlated action effects in decision theoretic regression. In Geiger and Shenoy [23], pages 30–37.

[8] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[9] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artif. Intell.*, 121(1-2):49–107, 2000.

[10] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *AAAI/IAAI, Vol. 2*, pages 1168–1175, 1996.

[11] Anthony Cassandra, Littman Michael, and Zhang Nevin. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the 13th Annual Conference on Uncertainty*

in Artificial Intelligence (UAI-97), pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[12] Anthony Rocco Cassandra. *Exact and approximate algorithms for partially observable markov decision processes*. PhD thesis, Brown University, 1998. Adviser-Leslie Pack Kaelbling.

[13] A. Davidson. Opponent modeling in poker: Learning and acting in a hostile environment. Master's thesis, University of Alberta, 2002.

[14] D. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming, 2001.

[15] E. D. de Jong. The maxsolve algorithm for coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005.

[16] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI/IAAI*, pages 106–111, 1997.

[17] Thomas Dean, Robert Givan, and Sonia M. Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In Geiger and Shenoy [23], pages 124–131.

[18] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 136–143, Washington, DC, USA, 2004. IEEE Computer Society.

[19] Z. Feng and E. Hansen. Approximate planning for factored pomdps. In *Sixth European Conference on Planning (ECP-01)*, 2001.

[20] Z. Feng and E. Hansen. An approach to state aggregation for pomdps. In *AAAI-04 Workshop on Learning and Planning in Markov Processes – Advances and Challenges*, pages 7–12. AAAI Press, 2004.

[21] Sevan G. Ficici and Jordan B. Pollack. A game-theoretic memory mechanism for coevolution. In *GECCO*, volume 2723 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2003.

[22] H. Fujita, Y. Matsuno, and S. Ishii. A reinforcement learning scheme for a multi-agent card game. In *IEEE International Conference System, Man. and Cybernetics (IEEE SMC '03)*, pages 4071–4078, 2003.

[23] Dan Geiger and Prakash P. Shenoy, editors. *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, August 1-3, 1997, Brown University, Providence, Rhode Island, USA*. Morgan Kaufmann, 1997.

[24] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artif. Intell.*, 147(1-2):163–223, 2003.

[25] Carlos Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University, August 2003. Adviser-Daphne Koller.

[26] Carlos Guestrin, Daphne Koller, and Ronald Parr. Solving factored POMDPs with linear value functions. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01) workshop on Planning under Uncertainty and Incomplete Information*, pages 67 – 75, Seattle, Washington, August 2001.

[27] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 709–715. AAAI Press / The MIT Press, 2004.

[28] Eric A. Hansen and Zhengzhu Feng. Dynamic programming for pomdps using a factored state representation. In *AIPS*, pages 130–139, 2000.

[29] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *UAI*, pages 279–288. Morgan Kaufmann, 1999.

[30] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[31] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[32] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Mach. Learn.*, 49(2-3):193–208, 2002.

[33] Kee-Eung Kim and Thomas Dean. Solving factored mdps using non-homogeneous partitions. *Artif. Intell.*, 147(1-2):225–251, 2003.

[34] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proc. of the 26th ACM Symposium on Theory of Computing (STOC)*, pages 750–759, 1994.

[35] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, 1997.

[36] Kevin B. Korb, Ann E. Nicholson, and Nathalie Jitnah. Bayesian poker. In *Proc. 15th Conference on Uncertainty in Articial Intelligence*, pages 343–350, 1999.

[37] H.W. Kuhn. *A Simplified Two-Person Poker*, volume 1 of *Contributions to the Theory of Games*. Princeton University Press, 1950.

[38] H.W. Kuhn. Extensive games and the problem of information. *Annals of Mathematics Studies*, 28:193–216, 1953.

[39] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1997.

[40] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[41] Andrew Y. Ng and Michael I. Jordan. Pegasus: A policy search method for large mdps and pomdps. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI*, pages 406–415. Morgan Kaufmann, 2000.

[42] Frans Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Best-response play in partially observable card games. In *Benelearn 2005: Proceedings of the 14th Annual Machine Learning Conference of Belgium and the Netherlands*, pages 45–50, February 2005.

[43] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of markov decision processses. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[44] Christos Papadimitriou. Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753, New York, NY, USA, 2001. ACM Press.

[45] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, (to appear).

[46] Pascal Poupart and Craig Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15*, pages 1547–1554, 2002.

[47] Pascal Poupart and Craig Boutilier. VDCBPI: an approximate scalable algorithm for large POMDPs. In *Advances in Neural Information Processing Systems 17*, pages 1081–1088, 2004.

[48] Rob Powers and Yoav Shoham. New criteria and a new algorithm for learning in multi-agent systems. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[49] M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[50] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[51] Jiefu Shi and Michael L. Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 333–345. Springer-Verlag, 2002.

[52] Yoav Shoham, Rob Powers, and Teg Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, Computer Science Department, Stanford University, 2003.

[53] E. J. Sondik. *The optimal control of partially observable Markov decision processes*. PhD thesis, Stanford University, 1971.

[54] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 2005, In press.

[55] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. Apricodd: Approximate policy construction using decision diagrams. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *NIPS*, pages 1089–1095. MIT Press, 2000.

[56] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[57] Gerald Tesauro. Practical issues in temporal difference learning. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 259–266. Morgan Kaufmann Publishers, Inc., 1992.

[58] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1947. 2nd edition.

[59] E. Zermelo. Uber eine anwendung der mengenlehre auf die theorie des schachspiels. In E. W. Hobson and A. E. H. Love, editors, *Proceedings of the Fifth International Congress of Mathematicians II*, pages 501–504. Cambridge University Press, 1913.